NOTICE: When government or other drawings, specifications or other data are used for any purpose other than in connection with a definitely related government procurement operation, the U. S. Government thereby incurs no responsibility, nor any obligation whatsoever; and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use or sell any patented invention that may in any way be related thereto.

# AIR FORCE INSTITUTE OF TECHNOLOGY

AIR UNIVERSITY
UNITED STATES AIR FORCE

## SCHOOL OF ENGINEERING

WRIGHT-PATTERSON AIR FORCE BASE, OHIO

THESIS

Presented to the Faculty of the School of Engineering of
the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

# THRESHOLD LOGIC

GE/EE/62-2

Noel Stewart Alton
Major USAF

Graduate Electrical Engineering

December 1962

## Preface

The selection of threshold logic as the subject for this thesis was due to my interest in modern mathematics, computer logic, and two specific suggestions: one, the suggestion by Captain F. M. Brown that non-linear flow-graph analysis might profitably be applied to networks of threshold elements; and two, the suggestion by Captain R. B. Stuart that an algorithm he developed for realizing threshold functions might constitute a necessary and sufficient test for the realizability of Boolean functions. Study of the extensive literature on threshold logic revealed that a large and important body of fact, theory, and synthesis procedures has been developed by the intensive research of the past few years; but in the process of this development, much confusing (and sometimes conflicting) notation and terminology has been generated. Since no textbook on threshold logic has yet appeared, it seemed desirable to present a brief unified description and survey of the field of threshold logic. This effort comprises Chapter I of this study.

The analysis of threshold gate networks has received only limited attention in the literature. Therefore, methods for accomplishing such analysis were developed and are presented in Chapter II. Many excellent synthesis procedures have been developed for realizing arbitrary Boolean functions in terms of threshold gates; however, these methods are generally difficult to apply to functions containing a large number of variables, unless the procedures are programmed for use on a digital computer. Hence a relatively simple method for synthesizing such Boolean functions directly from their algebraic expressions was developed and is presented in Chapter III.

Study of Captain Stuart's algorithm revealed that counter-examples could be constructed to disprove the conjecture that it provided a necessary and sufficient test for realizability of a Boolean function. The algorithm and a counter-example are presented in Chapter IV.

The bibliography, while extensive, is by no means exhaustive. However, the sources listed provide other references to important papers in this field.

This study assumes that the reader is familiar with Boolean algebra and conventional switching theory. Since truth tables and threshold gate realizations in symbolic form can be considered as additional ways to express Boolean and threshold functions, the convention of treating such representations as equations has been adopted. Hence, throughout this study these representations are numbered in sequence as equations rather than treating them as figures or tables. In the interest of further simplicity, inequalities have also been numbered in the same sequence as the equations.

It is a pleasure to acknowledge my heavy indebtedness to those persons whose assistance and guidance made this study possible: to Mr. D. J. Boaz and Captain Bill Wilson of the Electronic Technology Laboratory, Wright-Patterson Air Force Base, who willingly made available much of the important literature on threshold logic and with whom many fruitful discussions were held; to Captain Frank M. Brown, my thesis advisor, whose patient nature and exceptional teaching ability were severely strained in many invaluable consultations; and to my

wife, Barbara Lee, who has endured two years of "widowhood" with unfailing patience and understanding. Without her help and encouragement I would have found the path very rough indeed.

Noel Stewart Alton

# Contents

Page

## Abstract

Threshold logic is the body of fact, theory, and procedures pertaining to the properties and characteristics of threshold gates, threshold functions, and threshold gate networks. A brief but unified survey of the present "state of the art" is presented. Methods for accomplishing the logical analysis of both general threshold gate networks and symmetric threshold gate networks are developed. The effect of feedback on a simple threshold gate is analyzed; and a procedure for synthesizing an arbitrary Boolean function directly from its algebraic expression is given. A counter-example to a conjectured necessary and sufficient test for the realizability of a Boolean function is given.

# THRESHOLD LOGIC

## I. Description and Survey

The design of combinational networks in binary logic systems has historically been based on the use of gates embodying the elementary switching functions. These functions - AND, OR, NOT, NAND, and NOR - also have had the desirable property of being readily realized with mechanical and electronic devices. Developments in recent years, however, have produced intense interest in a type of switching device called a threshold element, or threshold gate. This gate is in many ways more complex and interesting than conventional gates; unlike these gates a threshold element can treat its inputs asymmetrically. Hence a single unit can generate switching functions far more complex than those generated by a conventional gate.

Extensive investigation of this new gate was due in large measure to the early work of Karnaugh (Ref 17), who pointed out the logical possibilities of a ferrite core, and to the early and widespread recognition of the essential similarities between the mathematical models of the threshold gate and the neuron. This similarity suggested the use of threshold gates as neuron-simulating elements in pattern-recognition and self-organizing machines. Intensive research over the past seven years has resulted in a body of fact, theory, and design procedures known as threshold logic. This logic, though not yet as unified and complete as conventional logic theory, now represents a most useful and general approach to logical design. Not least among its benefits is a major reduction in the number of gating devices needed to realize a complex switching function. This chapter presents some of the more important elements of threshold logic in sufficient detail, it is hoped, to serve as a useful introduction to this growing and important segment of computer technology. The notation presented in the following section will be used consistently throughout this report.

## Notation

The following logical symbols are used:

$\oplus$ : exclusive disjunction

$+$ : inclusive disjunction

$.$ : conjunction

$=$ : logical equivalence

$\equiv$ : identity

$\bar{A}$: complementation (negation) of A.

Conjunction is also shown by juxtaposition where this usage leads to no confusion. Logical truth and falsity are consistently mapped to 1 and 0, while recognizing that other assignments could be selected. A useful Boolean difference operation is defined, based on the concept of exclusive disjunction (Ref 4:487): The fact that

$$A \oplus \bar{A} = 1 \tag{1}$$

is extended by definition to the following difference operation:

$$A = 1 - \bar{A}$$

or (2)

$$\bar{A} = 1 - A$$

A Boolean function of n binary variables $x_1, x_2, \ldots, x_n$ is denoted by F where

$$F \equiv F(x_1, x_2, \ldots, x_n) \tag{3}$$

A threshold function of n binary variables $x_1, x_2, \ldots, x_n$ is denoted by T where

$$T \equiv T(x_1, x_2, \ldots, x_n) \tag{4}$$

The letter n denotes the number of arguments of F or T. The binary variables used as arguments of F or T are represented by the suitably subscripted lower case letters x or y, or by the capital letters A, B, C, . . . , excluding the letters F and T. The word "function" appearing without modifiers is intended to mean a logical function. The subscripts T and F on any symbol, i. e., $A_T$, or $A_F$, identify that symbol as a member of the respective true or false subset of the set of all such symbols. The lower case letters w and t, subscripted as required, represent, respectively, the weights and threshold value of a threshold gate; i. e., they are real constants which give the values assigned to the weights and threshold of the threshold gate. When the threshold is represented as a bias it is depicted by $w_o$ instead of t.

Use of the symbol $\sum$ is restricted to arithmetic summation; that is, it will not be used to indicate Boolean summation or disjunction. An indicated summation, $\sum_{i=1}^{n} w_i x_i$, implies that the n x's have been mapped to 0 or 1, prior to summation, by a particular assignment of truth values; hence the indicated summation represents a real number. An assignment of a particular set of truth values to the n variables of F is referred to as a valuation on the n variables of F. Use of the symbols > < or $\geq \leq$ is limited to conventional algebraic inequality; and logical implication is noted by the symbol → , i. e., if $F_1 \rightarrow F_2$ then $F_1$ implies $F_2$.

If, in a given context, the conventional algebraic operations of addition and multiplication can be confused with logical disjunction or conjunction, the correct interpretation will be noted. Certain essential definitions will now be given.

## Definitions

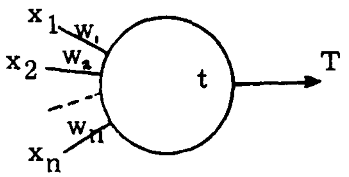
Boolean Function: A Boolean function is a member of the set of all switching or logical functions; it is sometimes called a "truth" function (Ref 19:1). Distinction is made between a Boolean function and a Boolean expression, since the Boolean function may be represented by a variety of Boolean expressions, no one of which may be said to be "the function." Two Boolean expressions are particularly useful in specifying a given switching or logical function: the canonical minterm form, whose terms correspond directly to those rows of the truth table for which the function is true; and a minimal normal disjunctive form in which F is expressed as a disjunction of essential prime implicants*, i. e., no prime implicant can be deleted without destroying the equivalence. This minimal form for F is sometimes called "the irredundant normal disjunctive form," abbreviated INDF (Ref 28:19). For simplicity, this study will use the term minimal normal form when

*A prime implicant of a function is a conjunction of variables that implies F and contains no shorter conjunction of variables also implying F.

referring to a Boolean function expressed in INDF. Several procedures exist to reduce a function to this minimal normal form, one of the more common being the Quine-McCluskey reduction method (Ref 24:288).

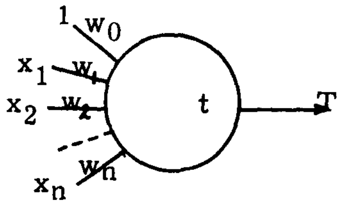
Threshold Gate. A threshold gate is an operator which maps the $2^n$ possible valuations on the n input variables to a binary output of truth or falsity, in such a way as to realize a particular subset of all Boolean functions. Members of this subset are called threshold functions. It performs this mapping operation by a process of linear summation of the weighted input values: if the weighted input sum is greater than the threshold level of the gate, an output representing truth is obtained; if the weighted input sum is less than the threshold level an output representing falsity is obtained.

Threshold gates are physically realized with many different devices, the most common of which are magnetic cores, resistor-transistor circuits, parametron circuits, resistor-tunnel diode circuits, and multiple coil relays (Ref 20:6). Although these circuit configurations differ widely, they are all represented logically by the following symbols:



(5a)

or alternatively,



(5b)

Note that in the alternate representation the threshold level is represented by a biasing input ( considered to have a truth valuation of 1) with weight $w_o = -t$. The representations are completely equivalent and both are used in the literature.

Threshold Function. Mathematically, threshold functions are defined as follows: A Boolean function F of n binary variables is said to be a threshold function if and only if there exists a set of n + 1 real constants $w_1, w_2, \ldots, w_n$ and t, such that, for all possible valuations on the n input variables of F, the following conditions hold:

$$\sum_{i=1}^{n} w_i x_i \geq t \qquad (6a)$$

for all valuations for which F is true; and

$$\sum_{i=1}^{n} w_i x_i < t \qquad (6b)$$

for all valuations for which F is false. This definition could alternatively be expressed in terms of the n + 1 real constants $w_o, w_1, w_2, \ldots, w_n$ and the inequalities would then be

$$w_o + \sum_{i=1}^{n} w_i x_i \geq 0 \qquad \text{if F is true,} \qquad (7a)$$

and

$$w_o + \sum_{i=1}^{n} w_i x_i < 0 \quad \text{if F is false.} \tag{7b}$$

The definitions are completely equivalent, differing only in the manner of expressing the threshold. Two important points should be noted:

a. These defining inequalities actually represent a set of $2^n$ inequalities corresponding to the $2^n$ possible valuations on the input variables. The inequalities express conditions which the n + 1 constants must satisfy if a given function is to be a threshold function.

b. Implicit in the definition is an ordering, or partial ordering, of the $2^n$ weighted input sums represented by $\sum_{i=1}^{n} w_i x_i$; that is, all such sums for which F is true must be greater than all such sums for which F is false. Stated another way, the smallest weighted true term(s) must be greater than the largest weighted false term(s). Thus, the threshold level t may have any value such that

$$\min\{\text{weighted true terms}\} \geq t > \max\{\text{weighted false terms}\} \tag{8a}$$

or

$$\min\{\text{weighted true terms}\} > t \geq \max\{\text{weighted false terms}\} \tag{8b}$$

Hence it is a matter of definition whether the weighted input sum equal to t is considered true or false. The interpretation of Ineq (8a) is used consistently throughout this study since it is the more common usage.

Threshold functions are known by many names in the extensive literature existing. Some of the more common terms are given below:

a. Realizable Function: A threshold function is a function realized by a threshold gate. Hence the term "realizable function" is used in the limited sense of being a function which can be realized by a single threshold gate. This is a commonly used term.

b. Setting Function (Ref 17:570): The inputs to a magnetic core are currents on several lines, each of which is associated with a given number of turns on the core. The resulting magnetomotive force is the weighted sum of the input signals (called the "input composite"), with the turns on the core representing the weights. If the input composite is sufficient to "set" the core, then an output pulse will be obtained during the next phase and the function which "set" the core is termed the setting function.

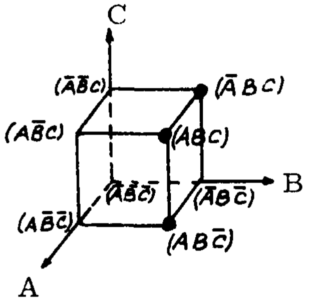
c. Linear Input Function (Ref 20:6): This term was derived from the manner in which the threshold gate performs its summation on the weighted inputs.

d. Linearly Separable Function (Ref 19:1): This name derives from geometric considerations discussed in the next definition. The term is widely used in the literature, frequently in abbreviated form LSF. The two expressions "realizable function" and "linearly separable function" will be used as synonymous terms for threshold functions in this study.

Geometric Representation. For a function of n variables there are $2^n$ possible assignments of truth values to the arguments of the function; and the particular valuations for which the function is true define the given function. The $2^n$ valuations can be mapped to the vertices of an n-dimensional unit hypercube; then those vertices corresponding to the valuations for which F is true are termed true vertices and constitute a geometric representation of the function. The remaining vertices correspond to those valuations for which F is false and are called false vertices. As an example, consider the function

$$F = B(A + C) = ABC + AB\bar{C} + \bar{A}BC \qquad (9)$$

This three variable function has the following geometric representation:



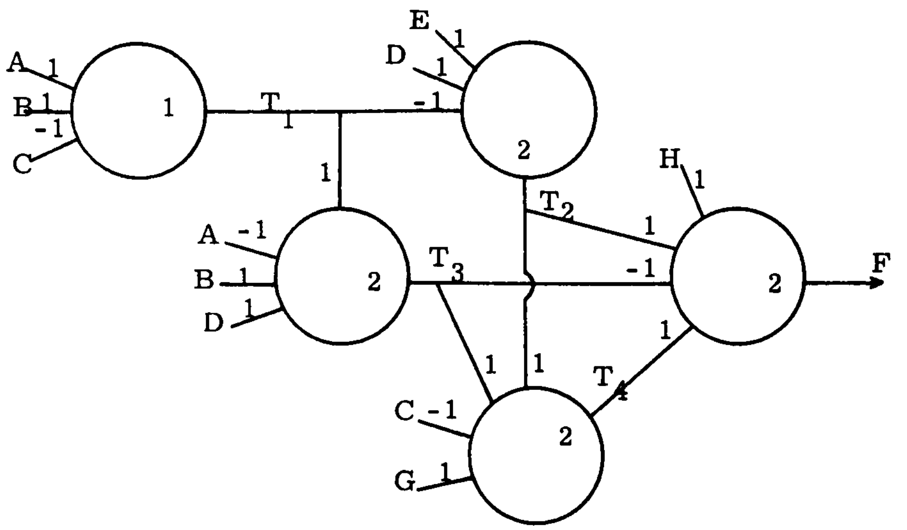
(10)

The three"dotted" vertices represent the function and hence are true vertices; all others are false vertices. While higher dimensionalities may be difficult to visualize, the concept is valuable and mathematically meaningful. Since threshold functions are a subset of Boolean functions, they can, of course, be represented geometrically. From their defining Ineqs (7) the following equality can be written for the particular weighted input sum which equals the threshold:

$$w_o + w_1x_1 + w_2x_2 + \ldots + w_nx_n = 0 \qquad (11)$$

Eq (11) represents the equation of a hyperplane in n-dimensional space, where $w_1, \ldots, w_n$ are proportional to the direction cosines of a normal to the plane, and $w_o$ is proportional to the distance of the hyperplane from the origin. This plane separates the true vertices from the false vertices of the n-dimensional cube. Thus the term "linearly separable" is a descriptive name for threshold functions. In Eq (10) note that the given function is linearly separable, since the three true vertices can be separated from the five false vertices by a two-dimensional plane. This geometric concept and terminology are widely used.

Threshold Network. A threshold network is a logic circuit composed of threshold elements. A simple network, discussed in Chapter II, is shown below:



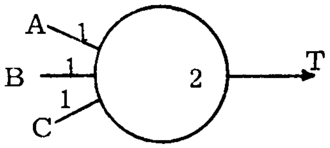
(12)

Threshold networks to perform many arithmetic functions are given in Ref 11:287. Others are discussed in later chapters.

Threshold Logic. Threshold logic is the body of fact, theory, and procedures pertaining to threshold gates, threshold functions, and networks of threshold gates. An interesting subdivision of threshold logic is termed majority logic. It is a system of logic based on an augmented Boolean algebra which includes a majority operator, #, defined by (Ref 9:17):

$$A \# B \# C = AB + AC + BC \tag{13}$$

Majority logic is applicable only to networks of three input majority decision elements. Such an element is a simple type of threshold gate:

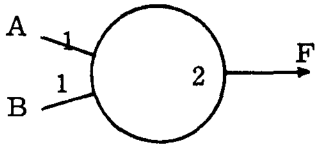


(14)

Although majority logic gates are not as flexible in logical design as more general threshold gates, the desire for a minimum number of different "building blocks" in a given computer design provides strong motivation for their use. Even though the term majority logic is commonly accepted to mean "two out of three" logic, the name is occasionally applied to more general majority decision studies; and it is also sometimes applied to the general study of threshold functions (Ref 21). Certain practical considerations will now be discussed before proceeding to a more detailed study of threshold functions.

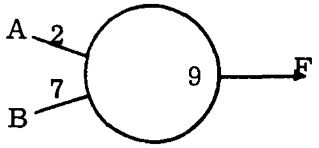
## General Considerations

Weight and Threshold Considerations. By definition, if a function is to be a threshold function, a set of n + 1 constants must be found which satisfy Ineqs (6) or (7). Such a set of n + 1 constants is not unique, since if any set is found that satisfies the defining inequalities, it may be multiplied or divided by any positive real number to generate a new set of n + 1 constants also satisfying the defining inequalities. Further, there may be more than one set of n + 1 constants, not linearly-related, which satisfy the required inequalities. As an elementary example, F = AB can be realized by



(15)

or



(16)

and many others. Note also that the definition of a threshold function only requires these n + 1 constants to be real numbers, not necessarily integers. However, they can be, and universally are, restricted to integer values with no loss of generality in the functions generated. This can be shown by the following reasoning: Assume a set of n + 1 rational constants which satisfy the defining inequalities; then this set can be multiplied through by a common denominator to obtain an integer realization. Assume now a set of n + 1 constants, some or all of which are irrational numbers, which satisfy the defining inequalities; then each irrational constant can be replaced by a rational number sufficiently close in value so that the threshold t may still be selected in the range required by Ineq (8a). Thus restricting the n + 1 real constants to integer values provides a desiiable simplification with no loss of generality (Ref 28:7).

Normalization of Inputs and Outputs. All previous discussion has implicitly assumed that the input and output values of a threshold gate are:

$$\left.\begin{array}{ll} \text{true input } x_i = 1 \ , & \text{false input } x_i = 0 \\ \text{true output } T = 1 \ , & \text{false output } T = 0 \end{array}\right\} \qquad (17)$$

Actually, of course, the inputs and outputs may be any two voltage levels, current levels, phases, or frequencies, depending on the physical realization. It has been shown, however, that any circuit using the normalized inputs and outputs given by Eqs (17) can be transformed by a linear transformation into one having more realistic input and output levels (Ref 20:8). Hence no loss of generality results from using the normalized values given in Eqs (17).

Physical Constraints. Many physical constraints exist and their effect must be taken into account during the final phase of logical design. Chief among these constraints is component tolerance which, when combined with the required tolerances in input and output signal levels, puts a definite bound on the gate's ability to discriminate between different weighted input sums. There are other design considerations, such as the fan-out or driving capability of the gate; the problem of even loading of the inputs; the problems imposed by physical dimensions, such as the number of turns on a ferrite core; the desire to obtain regular patterns of threshold gates for simplicity in manufacturing and servicing; and many others of a similar nature. These constraints will not be discussed further, not because they are not important, but because this study is principally concerned with purely logical considerations. An excellent discussion of the effects of constraints on logical design is contained in Ref 27 Ch 3.

Minimal Realization. The problem in conventional logic theory of what constitutes, and how to attain, a minimal realization of a switching function is also a problem in threshold logic. The concept of "minimal" realization may be approached from the points of view of component or circuit complexity, component accuracy (tolerance), or circuit delay. The interrelation between the three considerations is obvious, and in any given design problem the criteria for minimality are determined in terms of cost, permissable delay, and the function to be performed. In threshold logic studies it is usual to consider the minimal realization of a threshold function to be the smallest possible sum of weights plus threshold values which will realize the given function. In the realization of a general Boolean function, it is usual to consider the minimal realization to be the smallest number of threshold gates that can be used, each with a minimal sum of weights plus threshold. These criteria are used in this study.

## Properties of Threshold Functions

Threshold functions, as a subset of Boolean functions defined by a threshold gate, have been the object of intensive study in their own right. The objective of this investigation has been to discover those properties a Boolean function must have to be a threshold function. That the answer is not obvious may be inferred from the following equations:

$$F_1 = AB + CD \tag{18}$$

and

$$\begin{aligned} F_2 = {} & A(BC + (B + C)(D + E + F) + DE) \\ & + BC(D + EF) + (B + C)DEF \end{aligned} \tag{19}$$

Eq (18) is not a threshold function and hence needs more than one threshold gate (two, actually) for its realization. Eq (19) is a threshold function and therefore is realizable with a single threshold gate. Some of these properties of realizable functions will now be discussed.

The Number of Threshold Functions. It is a well known fact in switching theory that $2^{2^n}$ functions can be defined for the $2^n$ combinations of n binary variables. The question arises as to what portion of these $2^{2^n}$ functions are realizable. Interestingly, fourteen of the sixteen functions of two variables (all but the equivalence and exclusive-or functions) are realizable (Ref 25:213); but the percentage becomes vanishingly small as n increases. This fact is graphically depicted in Table I:

Table I

Number of Boolean Functions, Threshold Functions, and Symmetry Types as a Function of n

| No. of Arguments: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Boolean Functions | 2 | 4 | 16 | 256 | 65536 | $\sim 4.3 \times 10^9$ | $\sim 1.8 \times 10^{19}$ | $\sim 1.0 \times 10^{36}$ |
| Threshold Functions | 2 | 4 | 14 | 104 | 1882 | $\sim 9.4 \times 10^4$ | $\sim 1.5 \times 10^6$ | $\sim 1.0 \times 10^{10}$ |
| Symmetry Types Boolean Functions | 2 | 3 | 6 | 22 | 402 | $\sim 1.0 \times 10^6$ | $\sim 4.0 \times 10^{14}$ | -- |
| Symmetry Types Thres. Functions | 2 | 3 | 5 | 10 | 27 | 119 | 1113 | -- |

(From Ref 30:119)

The symmetric functions listed in the table are discussed below.

Symmetric Functions, Complements and Duals. Symmetric functions are Boolean functions which remain invariant under all permutations of the input variables. A simple example of a symmetric function is

$$F = ABC + \bar{A}BC + A\bar{B}C + AB\bar{C} \tag{20}$$

which may alternately be expressed as

$$F = AB + C(A\bar{B} + \bar{A}B) \tag{20a}$$

or

$$F = BC + A(B\bar{C} + \bar{B}C) \tag{20b}$$

or

$$F = AC + B(A\bar{C} + \bar{A}C) \tag{20c}$$

If any member of a symmetry class is a threshold function then all members of that class are threshold functions (Ref 20:9). This concept has been extended by proving that the property of linear separability remains invariant under repeated applications of the transformations of complementation and/or permutation of the variables of a threshold function (Ref 16: Ch 3). It has also been shown that duals and complements of a threshold function are themselves threshold functions. These useful facts are used in later discussions.

Threshold Function Characterizations. In general, a Boolean function has more than one minimal normal expression. A threshold function, however, has a unique minimal normal expression, and this expression contains all of the prime implicants of the function (Ref 27: 2. 36). Two other representations, in non-Boolean form, have been proven for threshold functions:

a. A threshold function can be characterized by $(w_1, w_2, \ldots, w_n; t)$ where the w's are the weights and t is the threshold of the threshold gate which realizes the function. In other words, these n + 1 constants define a unique threshold function T (Ref 27:2. 3). Recall, however, that the reverse implication does not hold: a given threshold function T does not define a unique set of n + 1 constants.

b. A threshold function of m minterms, expressed in positive unate form, can be characterized uniquely by the particular set of n + 1 real numbers, $\left[\sum 1(n), \sum 1(n-1), \ldots, \sum 1(2), \sum 1(1); m\right]$, where the indicated sums are column sums of the function's truth table, summed over the m true entries only (Ref 7). This is an interesting, but, thus far, not very useful representation. The meaning of the phrase "positive unate form" will be defined in the following section.

## Necessary and Sufficient Conditions for Realizability.

Despite the efforts of many authorities, the problem of finding an algebraic test for realizability, which may be applied directly to Boolean expressions, remains unsolved; and some authors now believe that such an algebraic test will never be found (Ref 30:33). The consistency of the set of $2^n$ inequalities defined by Ineqs (6) or (7) remains the only completely applicable necessary and sufficient test for the realizability of a function. Several properties, to be presented, have been defined which provide necessary conditions for realizability; but the sufficiency of these properties is limited to Boolean functions of relatively few arguments. Concise geometric conditions can be stated that are necessary and sufficient for a function to be linearly separable; but no convenient method of applying these conditions is available. Almost all of the algorithms and tests existing in the literature reduce, in one way or another, to a test for the consistency of the set of $2^n$ inequalities prescribed by the definition of a threshold function. Before discussing these tests, which are essentially synthesis procedures, certain properties which are necessary conditions for realizability will be discussed.

Unateness. A unate function is one that can be represented by a minimal normal expression in which no variable appears both complemented and uncomplemented (Ref 19:1). As examples, consider the functions

$$F_1 = AB + A\bar{C} \qquad (21)$$

and

$$F_2 = AB + \bar{A}\bar{B} \qquad (22)$$

$F_1$ is unate, but $F_2$ is not, since it cannot be reduced to any minimal normal form in which neither variable appears both complemented and uncomplemented. The concept of unateness is extended by another definition:

A function F is positive (negative) in an argument, represented by A, if there exists a minimal normal expression for F in which A appears only as an uncomplemented (complemented) variable. Thus F is unate in an argument if it is either positive or negative in that argument. Further, if F is positively (negatively) unate in each variable then F is termed a positive (negative) unate function. This concept is useful when synthesis procedures are discussed, since the weights necessary to realize a positive unate threshold function are always positive. The property of unateness is a necessary condition for linear separability of a function, but it is a sufficient condition only for functions with $n \leq 3$ (Ref 19:3). As an example, the function of four variables

$$F = AB + CD \tag{18}$$

is unate but it is not a threshold function.

Complete Monotonicity. A rather difficult concept of "complete monotonicity" of a function has been defined (Ref 28:11) as a stronger necessary condition for realizability than unateness. Certain notation and definitions must be presented before the definition of monotonicity can be given:

Consider a function F of n variables. Let X and Y be valuations on a subset of the n variables. As examples, if n = 5, let

$$X \equiv (x_1 = T,\ x_3 = T,\ x_4 = F) \tag{23}$$

Then $F_x$ is the restricted function

$$F_x \equiv F(1,\ x_2,\ 1,\ 0,\ x_5) \tag{24}$$

Similarly, if

$$Y \equiv (x_1 = T,\ x_2 = F,\ x_3 = F) \tag{25}$$

then $F_y$ is the restricted function

$$F_y \equiv F(1,\ 0,\ 0,\ x_4,\ x_5) \tag{26}$$

Further, two functions $F_1$ and $F_2$ are said to be comparable if one implies the other; that is, $F_1$ and $F_2$ are comparable if

$$\begin{array}{ll} & F_1 \rightarrow F_2 \\ \text{or} & \qquad\qquad\qquad (27) \\ & F_2 \rightarrow F_1 \end{array}$$

Complete monotonicity can now be defined:

A switching function F is completely monotonic when, for every two valuations X and Y on some common subset of F's arguments, $F_x$ and $F_y$ are comparable. Thus to check for complete monotonicity, it is necessary to compare all $2^n$ possible subsets of valuations with each other, checking for any implication between the two restricted functions $F_x$ and $F_y$ for each pair. Since this is obviously a formidable procedure the concept of complete monotonicity is decomposed into the concepts of 1-monotonicity, 2-monotonicity, . . . , n-monotonicity. The latter is synonymous with complete monotonicity. 1-monotonicity restricts the valuations X and Y to subsets of 1 variable; 2-monotonicity to subsets of 2 variables, and so on. Two additional theorems simplify the procedure for checking for complete monotonicity: the first proves that if a function is shown to be m-monotonic, then the only valuations that need be examined for determining m+ 1 monotonicity are X and Y where $Y \equiv \bar{X}$, instead of all possible pairs of valuations on the m + 1 variables. The second theorem proves that if a function is shown to be n/2 -monotonic then it is completely monotonic. Hence, if F, a function of six variables, is to be checked for realizability, it is necessary to check only for 1-, 2-, and 3-monotonicity of the function. If at any step the function is found not to be monotonic, then the given function is not realizable and the process is discontinued. It can be shown that the condition of 1-monotonicity is equivalent to the property of unateness; and 2-monotonicity is equivalent to a test named the "sieve method" (Ref 25:210) which was proposed as a possible necessary and sufficient test for realizability. Complete monotonicity comprises a necessary test for realizability, but it is a sufficient test only for $n \leq 6$. For $n \geq 9$, it has been proven not to be a sufficient test; its sufficiency for $n = 7$ or 8 has not been determined (Ref 30:59).

Geometric Conditions. Using convex set theory, several authors have proven that a necessary and sufficient condition for linear separability is that the function's true vertices and its false vertices form two disjoint convex hulls (Refs 13:225, 14:777). The latter reference tests for linear separability by applying a corollary: If there exists at least one point which is a convex combination of points from $S_T$ and which is also a convex combination of points from $S_F$ (where $S_T$ and $S_F$ are the convex hulls representing the respective sets of true and false vertices), then the function is not linearly separable. Using convex set theory, a set of $n + 2$ linear algebraic equations in m unknowns is established, where m is the number of points jointly shared by $S_T$ and $S_F$. If these equations have a non-negative solution then the convex hulls are not disjoint and the function is not linearly separable.

Another geometric condition conjectured to be necessary and sufficient for linear separability has been presented (Ref 22:1335). This condition is that no set of vertices of the n-cube forms a parallelogram (or sub-cube) such that true vertices form one diagonal pair and false vertices the other diagonal pair. The meaning of diagonal vertices is illustrated below:

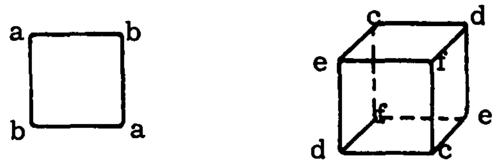


Fig. 1

Diagonal Vertices

This geometric condition has been shown to be analogous to complete monotonicity; it is therefore a necessary condition but sufficient only for functions with $n \leq 6$. Since there is no convenient method for determining if this condition is met ( at least for $n > 3$), utility of the procedure is limited.

Threshold Function Synthesis

A comprehensive coverage of the many synthesis procedures extant cannot be undertaken in this limited study. Rather, an attempt will be made to present the essential concepts of the major synthesis procedures, with references to the more important papers on synthesis given where appropriate. The synthesis of a threshold function will first be discussed, and then the procedures developed will be extended to the synthesis of arbitrary Boolean functions with networks of threshold elements.

It was previously noted that the definition of a threshold function establishes a set of $2^n$ inequalities on the $n + 1$ constants needed to realize it. Two facts are obvious:

a. The number of inequalities soon becomes unmanageable as n increases; and

b. a great number of redundant inequalities exists.

Hence, the essential element in most synthesis procedures is the method used to reduce the $2^n$ inequalities to a workable number. The reduced set of inequalities is then solved by various methods which include trial or iterative procedures and linear programming. In the absence of a suitable algebraic test to determine directly from the Boolean expression whether the given function is realizable, the synthesis methods proceed as if the function is indeed a threshold function. Then, if a set of weights and threshold can be generated, the synthesis procedure has served as a necessary and sufficient test to prove linear separability. If a set of weights and threshold cannot be found (i. e., the inequalities are inconsistent), the function is not realizable; and depending on the particular method being used, the procedure is either discontinued or extended to generate a multi-gate realization of the function.

Since it is convenient in most synthesis procedures to work only with positive unate functions, a simple method is presented which transforms a function to positive unate form for purposes of test and synthesis, and then converts the realization obtained for the transformed function to a realization for the original function:

The defining inequality for the weighted true terms of a function can be written

$$w_1x_1 + w_2x_2 + \dots + w_nx_n \geq t \qquad (28)$$

Assume that the given function F is positively unate in all but one variable, say $x_2$:

$$F \equiv F(x_1, \bar{x}_2, \ldots, x_n) \tag{29}$$

Assume now that the function is transformed to positive unate form by substituting $x_2$ wherever $\bar{x}_2$ appears in the expression, so that the transformed function is

$$F' \equiv F(x_1, x_2, \ldots, x_n) \tag{30}$$

Suppose that F' is realizable and that Eq (28) represents the set of weights and threshold that realize F': then the realization for the original function F can be obtained by substituting the quantity $(1 - x_2)$ for $x_2$ in the realization for F'. The resulting inequality will represent the realization of F:

$$w_1x_1 + w_2(1 - x_2) + \ldots + w_nx_n \geq t$$

or

$$w_1x_1 - w_2x_2 + \ldots + w_nx_n \geq t - w_2 \tag{31}$$

The basis for this procedure is, of course, the Boolean difference operation which has been defined. Thus any function can be transformed to positive unate form by changing all complemented variables to uncomplemented form. Then, when a realization is obtained for the transformed function F' it is changed to a realization for the original function F by:

a. changing the sign of the weight of each transformed variable,

and

b. adding the amount of the weights, for all variables transformed, to the original threshold value.

In all further discussions it will be assumed that the given function is transformed to positive unate form before applying any synthesis procedure.

Two primary methods are used to effect a reduction in the $2^n$ inequalities to be solved: ordering of the vertices and ordering of the weights. These procedures will now be explained.

Ordering of the Vertices. It has been pointed out that the $2^n$ possible valuations on the variables of a function can be mapped to the $2^n$ vertices of an n-cube. An ordering of these vertices will now be defined:

Let $X \equiv (a_1, \ldots, a_n)$ and $X' \equiv (a_1', \ldots, a_n')$ be two valuations on the n variables of F. Then by definition, $X < X'$ if and only if, for every i, $a_i \leq a_i'$. This ordering is a partial ordering only since some vertices are not comparable. Thus, (10001) < (10101) but (10001) and (01001) are not comparable. This partial ordering constitutes a lattice, and the vertices (00... 0) and (11... 1) are respectively the least vertex and the greatest vertex for positive unate functions. The defining inequalities for a threshold function require each weighted true term to be greater than each weighted false term. It is obvious that, in the partial ordering of the vertices defined above, the leastterms found will have the smallest weighted sums, and the greatest terms found will have the largest weighted sums. This provides a convenient method to reduce the $2^n$ inequalities to a more workable set: If the least true vertices are found and the greatest false vertices are found, then a reduced set of inequalities is obtained which consists of inequalities established between the weighted least true vertices and the weighted greatest false vertices. It is apparent that if this set of inequalities is consistent the entire set of $2^n$ inequalities is consistent, since the least true terms represent a smaller weighted sum than any other true term, and the greatest false terms represent a greater weighted sum than any other false term.

The least true terms can be found from the truth table for the given function by eliminating any true entry which "covers" another entry, i. e., (01011) "covers" (01001) and would be eliminated. True terms that finally remain are those not comparable and they constitute the set of least true terms. Similarly, the greatest false terms are obtained by eliminating any false entry in the truth table "covered" by another false entry. Thus (10001) is covered by (11001) and would be eliminated. The false terms remaining are those not comparable and constitute the set of greatest false terms.

A more convenient method of obtaining the least true set and greatest false set of vertices uses the minimal normal forms of the given function F and its complement $\overline{F}$. Recall that this form of a threshold function is unique and contains all of the prime implicants of the function. It has been shown (Refs 19, 27, 28) that the prime implicants of a positive unate function F represent the least true vertices of the function and the prime implicants of $\overline{F}$ represent the greatest false vertices of F. Thus, if we express F and $\overline{F}$ in minimal normal form we directly obtain the reduced set of inequalities between minimum true vertices and maximum false vertices. The following example synthesizes a given function and will make these procedures clear.

Example. Given the function (from Ref 19),

$$F = AB\overline{C} + B\overline{C}\overline{D} \tag{32}$$

1. Convert to positive unate form:

$$F' = ABC + BCD \tag{33}$$

2. F' is in minimal normal form; therefore, each prime implicant represents a least true vertex of F', or

$$\text{least true vertices} \equiv (1110),\ (0111) \tag{34}$$

3. $\overline{F}'$ is obtained through normal Boolean manipulation as

$$F' \equiv \bar{A}\bar{D} + \bar{B} + \bar{C} \tag{35}$$

and the greatest false vertices corresponding to the three prime implicants of $\overline{F}'$ are as follows:

$$\text{greatest false vertices} \equiv (0110),\ (1011),\ (1101) \tag{36}$$

4. Since each of the weighted least true terms of Eq (34) must be greater than each of the weighted greatest false terms of Eq (36) we can set up the following set of inequalities:

$$w_1 + w_2 + w_3 > w_2 + w_3 \tag{37}$$

$$w_1 + w_2 + w_3 > w_1 + w_3 + w_4 \tag{38}$$

$$w_1 + w_2 + w_3 > w_1 + w_2 + w_4 \tag{39}$$

$$w_2 + w_3 + w_4 > w_2 + w_3 \tag{40}$$

$$w_2 + w_3 + w_4 > w_1 + w_3 + w_4 \tag{41}$$

$$w_2 + w_3 + w_4 > w_1 + w_2 + w_4 \tag{42}$$

Thus the original $2^n$ inequalities have been reduced to a set of six inequalities, two of which are obviously redundant. If this reduced set is consistent, then the entire set of $2^n$ inequalities is necessarily consistent.

5. The above inequalities reduce to the requirement that $w_2$ and $w_3$ each be greater than $w_1$ or $w_4$. If we choose

$$w_1 = w_4 = 1 \tag{43}$$

and

$$w_2 = w_3 = 2 \tag{44}$$

the inequalities are satisfied.

6. Since the threshold t must satisfy Ineq (8a), we have, using Ineq (37),

$$1(x_1) + 2(x_2) + 2(x_3) + 1(\overline{x}_4) \geq t' > 1(\overline{x}_1) + 2(x_2) + 2(x_3) + 1(\overline{x}_4) \tag{45}$$

or

$$1(1) + 2(1) + 2(1) + 1(0) \geq t' > 1(0) + 2(1) + 2(1) + 1(0) \tag{46}$$

or

$$5 \geq t' > 4 \tag{47}$$

Therefore, the value for t' is selected as 5 and F' is realized by (1, 2, 2, 1;5). Note that any other of the reduced set of inequalities could have been used in place of Ineq (37) to determine t'. To obtain the realization for F we apply the rules previously given: Change the signs of those weights corresponding to transformed variables and add these weights to t'. Therefore, since $\overline{C}$ and $\overline{D}$ were transformed,

$$w_3 = -2 \tag{48}$$

$$w_4 = -1 \tag{49}$$

and

$$t = 5 - 2 - 1 = 2 \tag{50}$$

and F is realized by (1, 2, -2, -1;2).

It should be noted that the procedure was applied to the given function just as if it were known to be a threshold function; had this assumption been false it would have been so indicated by failure to obtain a solution of the reduced set of inequalities. This method of ordering of the vertices was first presented by McNaughton (Ref 19). It is essentially the method used in synthesis procedures presented in Refs 12, 27, 28, and 30. For many functions the reduced set of inequalities obtained may still be uncomfortably large. This set can be reduced still further by a procedure which determines the relative order of magnitudes of the weights needed to realize the function. A discussion of this method follows.

Ordering of the Weights. It has been proven, in many different forms (Refs 5, 12, 16, 27, 28), that the relative magnitudes of the weights needed to realize a threshold function T can be determined from the number of appearances of each variable in certain of the Boolean expressions for T. The canonical minterm form and the minimal normal form are generally used. Using first the canonical minterm form, the procedure for obtaining the relative ordering of the weights is as follows:

The relative ordering of the weights assigned to the n variables of T can be determined by counting the number of appearances of each variable in uncomplemented form in the minterms of T; the relative ordering of the number of appearances of each uncomplemented variable gives the relative ordering of the respective weights. The procedure is applied to any given function, again assuming the function to be a threshold function; and the truth or falsity of this assumption will be demonstrated by the ultimate success or failure in solving the reduced set of inequalities.

Example. Given the function

$$F = ABCDE + \bar{A}BC\bar{D}\bar{E} + A\bar{B}\bar{C}DE + A\bar{B}C\bar{D}E + \bar{A}\bar{B}C\bar{D}\bar{E} \qquad (51)$$

Counting the appearances of the uncomplemented variables, C appears 4 times, A and E appear 3 times, B and D appear 2 times: The ordering of the weights is therefore

$$w_c > w_a = w_e > w_b = w_d \qquad (52)$$

and this ordering can now be applied to the reduced set of inequalities obtained by ordering of the vertices to effect a further reduction in their number.

Since the number of minterms may be quite large for a function of many variables, it is often more convenient to obtain this ordering from the minimal normal form for F. Recall that the smaller the number of variables in a prime implicant, the greater is the "cover" of that term: if a prime implicant consists of a single variable, then $2^{n-1}$ of the minterms of F will contain that variable; if a prime implicant consists of two variables, the two variables will appear in $2^{n-2}$

of the minterms of F, and so on. Thus the relative ordering of the variables, and hence of the weights, can be obtained as follows:

a. Count the appearances of the variables in those prime implicants consisting of the least number of variables. Order the variables appearing in these "least" prime implicants in order of the number of appearances of each variable. This ordering of variables is of higher order than those obtained in successive steps.

b. Count the appearances of the variables in those prime implicants consisting of the next greater number of variables. The number of appearances of any variable already ordered is immaterial unless an equality exists in the ordering of step 1; if this is the case, then the number of appearances of these variables in step 2 may resolve the equality of ordering obtained in step 1. Order the variables not previously ordered. This ordering of variables is of lower order than the preceding ordering.

c. Continue in like manner until all variables are ordered. An example will clarify this procedure.

Example. Given the following function, expressed in minimal normal form:

$$F = A + BC + BD + CE + BEFG + EFGH \qquad (53)$$

1. Prime implicants of 1 variable: A

2. Prime implicants of 2 variables: B and C appear twice, D and E appear once; therefore, the ordering obtained thus far is

$$A > B = C > D = E \qquad (54)$$

3. Prime implicants of four variables: E, F, and G appear twice, B and H appear once; the equalities of Ineq (54) can now be resolved as $B > C$ and $E > D$. The ordering obtained for F, G, and H is $F = G > H$, and the overall ordering is

$$A > B > C > E > D > F = G > H \tag{55}$$

This ordering can now be used to further reduce the set of inequalities obtained by ordering of the vertices.

When the ordering of weights yields an equal ordering of two weights, such as F and G in the previous example, common practice assigns equal weights to these variables. An interesting fact proved by Winder (Ref 30:87) is that an assignment of equal weights in these cases does not necessarily provide a minimal realization of the function.

Other Methods for Reducing the Set of Inequalities. Other methods exist either to obtain an initial reduced set of inequalities or to effect still further reductions in the reduced set obtained after applying the two procedures given above. Coates and Lewis (8:447) present a rather complex method of decomposing a given function into a function "tree". The method utilizes any symmetry of the function in subsets of the arguments: it successively factors out the highest order groups of symmetric variables and determines the next level of symmetry. The process is continued until it cannot be carried further. Equal weights are assigned to all variables in a given symmetry group and the highest order symmetric group of variables is given the highest weight. A reduced set of inequalities is obtained by the method of factoring which forms a function "tree." The relative order of the weights of the symmetry groups is used to reduce this basic set still further; the resulting reduced set is solved by a complicated process termed "reconstruction."

Winder (Refs 28, 29, 30) uses the results of previous checks for monotonicities to deduce certain "reducing" relations. These are used to effect still further reductions in the set of inequalities obtained through ordering of the vertices and ordering of the weights.

Other Synthesis Methods. Numerous synthesis procedures have been devised which do not utilize the reducing procedures presented above. Linear programming techniques are used in several well-known procedures: Minnick (Ref 20:6) presents a synthesis procedure for symmetric functions which is applied to the entire set of $2^n$ inequalities and utilizes the Simplex algorithm to obtain a solution; Einhorn (Ref 10:615) presents a similar procedure for general Boolean functions and again utilizes the Simplex algorithm to obtain a solution. Akers (Ref 2) presents an interesting variation on the linear programming technique by showing that the solution of the $2^n$ inequalities may be viewed as a two-person, zero-sum game, and he utilizes game theory to obtain a solution.

Two synthesis procedures are contained in Ref 27 which differ from any of the above methods. The first of these is an iterative geometric test procedure which may have to be continued through $2^n$ iterations before determining if a function is linearly separable. The second method utilizes a class of orthogonal digital functions called Walsh functions, and a sub-class of these known as Rademacher functions.

### General Synthesis Procedures

The general problem of threshold logic synthesis is, of course, to obtain a realization of an arbitrary Boolean function in terms of a network of threshold gates. Various procedures to accomplish this synthesis are now reviewed.

Primitive Synthesis. Since the elementary AND and OR functions are simple threshold functions, any Boolean function can be given a two-level realization. Such a realization is obtained with the least number of gates if the function is first put in minimal normal form. Then each prime implicant is realized by an AND gate, and a single OR gate terminates the outputs from these gates. This is a minimal-time realization, but, in general, requires an excessive number of gates. This procedure is further discussed in Chapter III where a new synthesis method for multilevel realization of a function is presented.

Linear Programming. The linear programming techniques previously mentioned provide a means to accomplish two-level synthesis. Initial application of the linear programming method determines if the given function is linearly separable. If it is not, the method yields a "best" solution and a residue function. The procedure is reapplied to this residue function; again this function is shown to be linearly separable, or another "best" solution and another residue function are generated. The method is successively applied to the residues until the process terminates in a threshold realization of a residue function and no new residue. An advantage of this procedure is that it may be programmed on a digital computer.

Geometric Methods. The iterative geometric procedure for realizing threshold functions which was discussed on p 33 is also applicable to multilevel synthesis. Successive applications of the method yield successive linearly separable functions, which are each the sum of all previously generated linearly separable functions plus at least one additional true vertex of the n-cube. The process is continued until all true vertices have been included in at least one realizable sub-function.

Winder (Ref 30) presents a method based on an involved geometric decomposition of a function into linearly separable sub-functions. An important result of his method is that if a function can be realized with two threshold gates, this realization is guaranteed to be minimal.

Majority Logic Synthesis. Several procedures have been presented for multilevel synthesis using three-input majority gates. Cohn and Lindaman (Ref 9) present a direct synthesis method using majority logic to build a "tree" realization of the given function. Akers (Ref 3) gives a "truth table" method for such synthesis which can handle "don't care" conditions as well as fully-specified functions. A method of factoring Boolean algebraic expressions so as to effect a 3-input majority gate realization of a function is contained in Ref 27.

Synthesis from Boolean Expression. A method is given in Ref 27 for synthesizing a function, using 2-, 3-, 4-, or 5-input threshold gates, directly from the Boolean expression. This method factors the function (initially expressed in minimal normal form); then, using a table of standard reference form realizations, it assembles a network according to the factorization. The method presented in Chapter III of this study is a more general and useful method of synthesizing a function directly from its Boolean expression, since it can utilize gates with any number of inputs and does not require any auxiliary table of reference functions.

Symmetric Function Synthesis. In addition to Minnick's linear programming method for synthesizing symmetric functions which has been discussed previously, there is a procedure presented by Kautz (Ref 18:371) for synthesizing symmetric functions. It is based on the number of "transitions" undergone by any threshold gate, where a transition is defined to be a change in effective value of the threshold of an element brought about by the weighted input from a previous threshold gate. Algebraic equations based on the number of transitions necessary to realize a function are set up, and their solution yields a realization.

## Summary

Threshold logic can be divided, somewhat arbitrarily, into the following major sub-areas of concern: properties of threshold functions; necessary and sufficient tests for the realizability of Boolean functions; synthesis procedures for threshold functions; and synthesis procedures for general Boolean functions. This chapter has briefly discussed each of these major sub-areas in sufficient detail, it is hoped, to provide a basis for further study in the field of threshold logic.

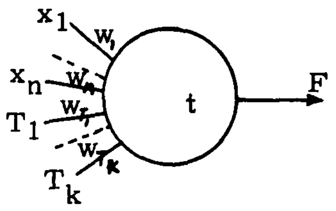
Methods for the analysis of threshold networks will be presented in the following chapter.

# II. Logical Analysis of Threshold Networks

Since the physical realization of a threshold network contains threshold gates, bias and synchronizing circuits, and power supplies, its conventional (electrical) network analysis is similar to that required for conventional computer circuits using the more common gating devices. The logical analysis of threshold networks is more complex, however, since the conditions (excitation of various inputs) under which a signal appears at the output of any gate is not as easily determined as it is in conventional logic circuits. The problem of accomplishing such logical analysis of threshold networks has received only limited attention in the literature. This chapter develops effective methods for analysis of general threshold networks and symmetric threshold gate networks, both without feedback loops. The effects of feedback on the action of individual threshold gates is also analyzed.

## Theory

As is the case with conventional logic networks, a threshold net containing no feedback loops uniquely determines a Boolean function of the n input variables. This Boolean function is a combination (in a manner defined by the given network) of all of the individual threshold functions uniquely defined by the individual threshold gates. Uniqueness of the overall network Boolean function follows from consideration of the uniqueness of a threshold function realized by a threshold gate: The final output of any threshold network is always the output from a threshold gate whose inputs are, in general, a combination of individual variables and other threshold functions, $T_1$, $T_2, \dots,$ $T_k$, all of which are unique outputs of other threshold gates:

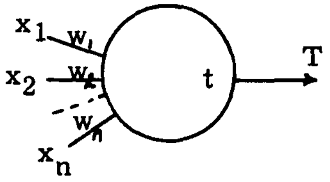


(56)

Thus, uniqueness of the overall function F is assured. Once the individual threshold functions are determined, the overall function F can be obtained. Hence, analysis of a single threshold gate is the first step in logical threshold net analysis.

Analysis of a Single Threshold Gate

Consider the general representation of a threshold gate with n inputs:



(57)

An output signal T appears whenever any combinatio of input leads is activated such that

$$\sum_{i=1}^{n} w_i x_i \geq t \, , \quad x_i = 1 \text{ or } 0 \qquad (58)$$

Theoretically, to determine the threshold function realized by the gate all $2^n$ possible input combinations must be examined. In practice, however, this number of input combinations is greatly reduced by the following analysis.

Consider a Boolean function F expressed in minimal normal form:

Assume that F has as its first three terms the following:

$$F = x_1 + x_2x_3 + x_4x_5x_6 + \ldots \tag{59}$$

If the $2^n$ possible input combinations are represented by the $2^n$ vertices of the n-dimensional hypercube, then all vertices containing $x_1$ are true vertices; and since exactly half of all of the vertices contain $x_1$ (the other half containing $\overline{x}_1$), F is true for these $2^n/2$ or $2^{n-1}$ vertices. Now consider the second term of F, a conjunction of two variables: one-fourth of the vertices contain $x_2x_3$, with the other three-fourths of the vertices containing either $\overline{x}_2x_3$, $x_2\overline{x}_3$, or $\overline{x}_2\overline{x}_3$; and F is again true for those vertices containing $x_2x_3$. Consider the third term, a conjunction of three variables: one-eighth of the vertices contain $x_4x_5x_6$, with the other seven-eighths containing one of the other seven possible combinations of complemented and uncomplemented variables. It is apparent that the smaller the number of variables appearing in a prime implicant the greater the "cover" (number of true vertices) represented by that term. This well-known fact suggests a method of reducing the number of input combinations that needs to be examined in determining the threshold function generated by a particular element.

Consider the threshold gate realized in Eq (57). If any single weight $w_i$ is such that

$$w_i \geq t \tag{60}$$

then the variable $x_i$ corresponding to $w_i$ is a prime implicant of T, and the number of input combinations to be examined is reduced to $2^{n-1}$. This means that $x_i$ need not be included in any combination of two or more variables to be compared with t. If the sum of any two weights, $w_j$ and $w_k$, is such that

$$w_j + w_k \geq t \quad \text{(algebraic addition)} \tag{61}$$

then $x_j x_k$ is also a prime implicant of T, and the number of input combinations to be examined is further reduced. The combination $x_j x_k$ is not included in any combination of three or more variables remaining to be compared with t. This procedure is continued until no further prime implicants can be formed, and the function F generated by the threshold gate is then the disjunction of these prime implicants. The procedure is systematized below.

Procedure. 1. Convert all negative weights to positive weights, since negative weights tend to complicate the analysis. This is accomplished, as explained in Chapter I, by

a. changing the sign of the weight from - to + ;

b. complementing the variable; and

c. adding an amount equal to the weight to the threshold.

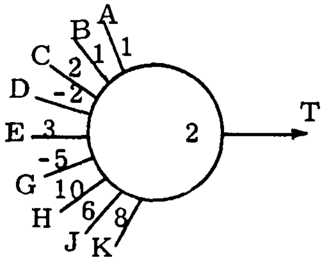
2. Compare each weight with the threshold t. List all variables corresponding to a $w_i \geq t$ as prime implicants of T and exclude these variables from any combination of two or more variables remaining to be compared to t.

3. Compare each possible pair of the remaining weights to

determine those pairs of variables with a sum of weights equal to or greater than t. List these pairs of variables as prime implicants of T and exclude these pairs from any combinations used in further comparisons.

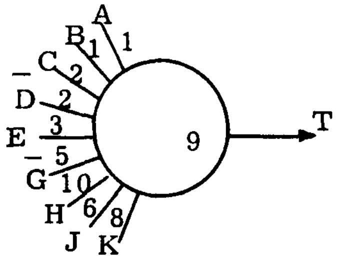
4. Continue this process for combinations of three, four, ... of the weights until no further prime implicants can be formed. Then T is the disjunction of all prime implicants found.

Example. Given the threshold gate shown. Determine the threshold function it realizes:



(62)

1. Convert all negative weights to positive weights:



(63)

2. Check for $w_i$, $w_j + w_k$, ... $\geq$ t:

| Combinations | Prime Implicants | |
|---|---|---|
| a. singles $\geq$ t: | H | (64) |
| b. pairs $\geq$ t: | AK, BK, CK, $\bar{D}$K, EK, $\bar{G}$K, JK, EJ, $\bar{G}$J | |

$$= K(A + B + C + \bar{D} + E + \bar{G} + J) + J(E + \bar{G}) \quad (65)$$

| Combinations | Prime Implicants |
|---|---|
| c. triples $\geq$ t: | $ACJ$, $BCJ$, $A\bar{D}J$, $B\bar{D}J$, $C\bar{D}J$, $AE\bar{G}$, $BE\bar{G}$, $CE\bar{G}$, $C\bar{D}\bar{G}$, $\bar{D}E\bar{G}$. |

$$\begin{aligned} &= J(AC + BC + A\bar{D} + B\bar{D} + C\bar{D}) \\ &\quad + \bar{G}(AE + BE + CE + C\bar{D} + \bar{D}E). \end{aligned} \tag{66}$$

d. quadruples $\geq$ t: $ABC\bar{G}$, $AB\bar{D}\bar{G}$

$$= G(ABC + AB\bar{D}) \tag{67}$$

e. quintuples $\geq$ t: $ABC\bar{D}E$ (68)

No further combinations can be formed which represent additional prime implicants. Therefore,

$$\begin{aligned} T &= H + K(A + B + C + \bar{D} + E + \bar{G} + J) + J(E + \bar{G}) \\ &\quad + J(AC + BC + A\bar{D} + B\bar{D} + C\bar{D}) \\ &\quad + \bar{G}(ABC + AB\bar{D} + AE + BE + CE + C\bar{D} + \bar{D}E) + ABC\bar{D}E \end{aligned} \tag{69}$$

or

$$\begin{aligned} T &= H + K(A + B + C + \bar{D} + E + \bar{G} + J) \\ &\quad + J(E + \bar{G} + (A + B)(C + \bar{D}) + C\bar{D}) \\ &\quad + \bar{G}(AB(C + \bar{D}) + E(A + B + C + \bar{D}) + C\bar{D}) + ABC\bar{D}E \end{aligned} \tag{70}$$

It should be noted that simple threshold functions such as AND or OR, and others for which n is small, do not require the procedure given above since they can be determined by inspection.

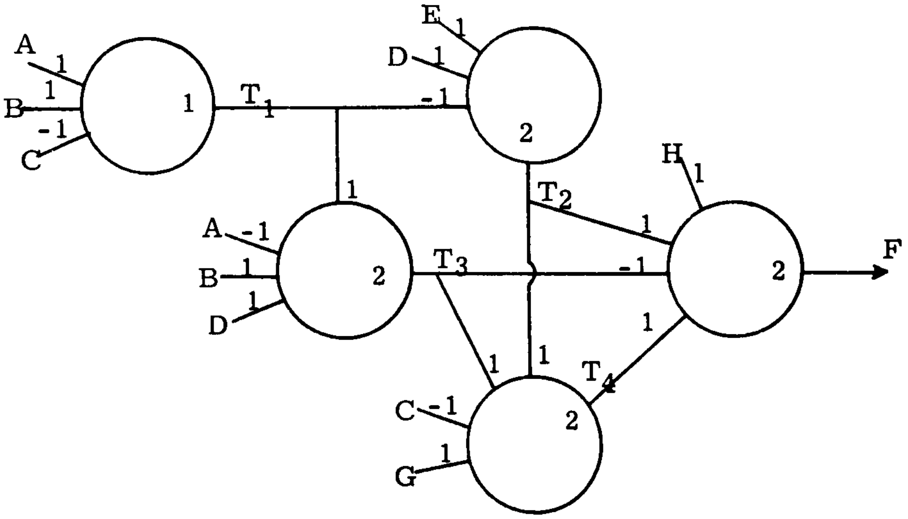
## Determination of the Overall Network Function

Once the threshold function for each threshold gate in the network has been obtained, determination of the function generated by the network is then a matter of successive substitution. The threshold function for the output gate will, in general, be in the form

$$F = F(T_i, T_j, \ldots, x_k, \ldots) \tag{71}$$

By substituting the expressions for $T_i$, $T_j$, ... into this equation the function, expressed in terms of the input variables only, is constructed. This procedure is best illustrated by use of an example.

Example. Given the threshold network shown. Determine the function F:



(72)

The individual threshold functions are found by inspection in this simple example:

1. $T_1 = \bar{C}(A + B)$ (73)

2. $T_2 = \bar{T}_1 DE = DE\overline{(A\bar{C} + B\bar{C})}$

which simplifies to

$$T_2 = DE(C + \bar{A}\bar{B}) \tag{74}$$

3. $$T_3 = \bar{A}(T_1(B + D) + BD) + ABDT_1$$
$$= T_1(B(\bar{A} + D) + \bar{A}D) + \bar{A}BD$$
$$= \bar{C}(A + B)(B(\bar{A} + D) + \bar{A}D) + \bar{A}BD$$

which simplifies to

$$T_3 = B(\bar{A}(\bar{C} + D) + \bar{C}D) \qquad (75)$$

4. $$T_4 = \bar{C}(GT_2 + GT_3 + T_2T_3) + CGT_2T_3$$
$$= T_2T_3(\bar{C} + G) + (T_2 + T_3)\bar{C}G$$
$$= BDE(\bar{A}\bar{B} + C)(\bar{A}\bar{C} + \bar{A}D + \bar{C}D)(\bar{C} + G)$$
$$+ \bar{C}G(CDE + \bar{A}\bar{B}DE + \bar{A}B\bar{C} + \bar{A}BD + B\bar{C}D)$$

which simplifies to

$$T_4 = BG(\bar{A}(\bar{C} + DE) + \bar{C}D) \qquad (76)$$

5. $$F = \bar{T}_3(H(T_2 + T_4) + T_2T_4) + HT_2T_3T_4$$
$$= HT_2\bar{T}_3 + HT_4\bar{T}_3 + T_2T_4(H + \bar{T}_3) \qquad (77)$$

and

$$\bar{T}_3 = \overline{B(\bar{A}(\bar{C} + D) + \bar{C}D)}$$

which simplifies to

$$\bar{T}_3 = A + \bar{B} + C\bar{D} \qquad (78)$$

Therefore, F can now be found in terms of the variables by substituting Eqs (74), (75), (76), and (78) into Eq (77):

$$F = H(A + \bar{B} + C\bar{D})DE(C + \bar{A}\bar{B}) + H(A + \bar{B} + C\bar{D})BG(\bar{A}(\bar{C} + DE) + \bar{C}D)$$
$$+ DE(C + \bar{A}\bar{B})BG(\bar{A}(\bar{C} + DE) + \bar{C}D)(H + A + \bar{B} + C\bar{D})$$

which reduces to

$$F = H(DE(C(A + \bar{B} + G) + \overline{AB} + ABG) + AB\bar{C}DG) \qquad (79)$$

This example brings out several important points:

a. The "flow" of the analysis is generally from those threshold gates with inputs consisting of variables only to the final output gate.

b. The procedure is simplified if substitution of previously determined threshold functions is accomplished as necessary in the determination of succeeding threshold functions; i. e., in the above example $T_1$ is substituted in the expressions for $T_2$ and $T_3$, and $T_2$ and $T_3$ are substituted into the expression for $T_4$. Simplification at each step greatly reduces the complexity of the final determination.

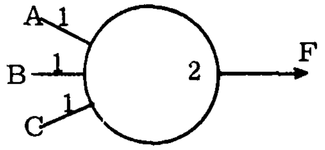
The method of analysis just presented is practical, easy to apply, and applicable to most threshold networks. There are two special classes of networks, however, which require further discussion: these are networks of threshold gates which generate symmetric threshold functions and networks containing feedback loops.

## Symmetric Threshold Gate Network Analysis

Symmetric threshold gates will be defined as any threshold gate which generates a symmetric threshold function. Symmetric functions have previously been defined (Chapter I) as Boolean functions which remain invariant to all permutations of the input variables; and it should be noted that a result of this definition is that a symmetric function of n variables is described uniquely by stating its truth value for k true inputs, with k = 0, 1, ..., n. The simple example previously given,

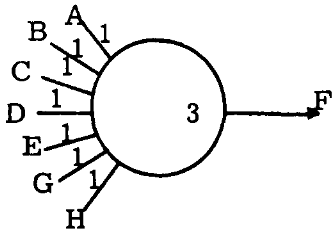
$$F = ABC + \bar{A}BC + A\bar{B}C + AB\bar{C} \qquad (20)$$

is uniquely described by specifying that F is true for two true inputs; it has the following realization:



(80)

Now consider the threshold gate shown:



(81)

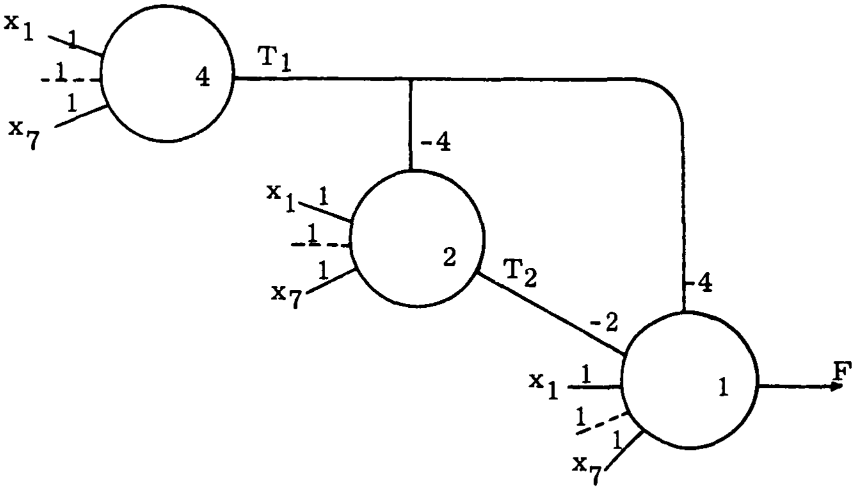
The threshold function realized by this gate can be written, but the unwieldiness of the expression can be demonstrated by recalling that the number of combinations of n things taken r at a time is

$${}_{n}C_{r} = \frac{n!}{r!(n - r)!} \qquad (82)$$

In this case the expression for the function would have thirty-five terms in its normal form. Thus a practical difficulty arises in applying the previous method of analysis to networks of symmetric threshold gates: the expressions obtained soon become so cumbersome as to defy analysis.

Other methods must be used, therefore, to analyze such networks. For the special case of networks of simple majority gates (two out of three), the analysis may make use of majority logic. Since majority logic requires specialized knowledge of Boolean algebra which has been extended to include the majority operator #, this case will not be further discussed. An algebraic method of analyzing networks which realize symmetric Boolean functions has been presented by Kautz (Ref 18:371). His method uses the concept of "transitions" which was explained previously (Chapter I p 36). The "truth table" method to be developed below is an alternate and somewhat simpler method of accomplishing such analysis. This method avoids formulating the Boolean expression for the function, relying instead on the truth table representation as the definition of the function realized.

General Considerations. Consider the following network (from Ref 19), which is typical of threshold networks used to realize symmetric functions:



(83)

Characteristics of networks such as this are as follows:

a. The input vector $(x_1, x_2, \ldots, x_n)$ is applied to each threshold gate and each input variable is assigned a unit weight. The leads for the input vectors are termed "input vector leads" and the input leads representing outputs from other gates are termed "threshold leads."

b. A valuation on the n variables of the input vector is expressed by an integer which represents the number of variables having a truth value of 1. These valuations are denoted by N in this development, where $0 \leq N \leq n$. Since each variable has a weight of 1, N also represents the weighted input sum for the input vector leads. Thus if N = 3, three input variables have a truth value of 1, and the weighted vector input sum is 3.

c. These networks are "feed forward" networks: the output of each gate is fed to one or more succeeding gates until the output gate terminates the process. The output gate is termed the lowest level gate, and the gate having input vector leads only (the input gate) is the highest level gate.

d. The defining inequalities for a threshold function, given by Ineqs (6), can be written in the following simple form for those symmetric threshold gates having, as additional inputs, the output(s) from higher level gates: For all possible values of N, the following conditions must hold:

$$w_{T_i}T_i + w_{T_j}T_j + \ldots + N \geq t \quad \text{whenever F is true} \qquad (84a)$$

and

$$w_{T_i}T_i + w_{T_j}T_j + \ldots + N < t \quad \text{whenever F is false} \qquad (84b)$$

$T_i$ and $T_j$ are used as binary variables in Ineqs (84). Since there are only n + 1 possible valuations on the input variables for these symmetric threshold gates, Ineqs (84) can be easily used to determine the output truth value for any threshold gate in the network. This fact provides the basis for the simple "truth table" method for determining the network function. The procedure follows:

Procedure and Example. Form a "truth table" for the network function with column headings as follows: The first column heading is N; succeeding column headings are the outputs from each threshold gate, ordered from the highest level gate to the lowest level gate. Thus the network function F is represented by the last column of the truth table. The first column is the vector (0, 1, 2,. . . , n) which represents the possible values of N. Applying this step to the network of Eq (83) we obtain the following:

| N | $T_1$ | $T_2$ | F |
|---|---|---|---|
| 0 | | | |
| 1 | | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |
| 6 | | | |
| 7 | | | |

(85)

2. By inspection of each threshold gate, set up the conventional algebraic inequality given by Ineq (83a) for each gate. For the example

of Eq (83) we have the following inequalities:

a. $T_1$ is true whenever

$$N \geq 4 \tag{86}$$

b. $T_2$ is true whenever

$$-4T_1 + N \geq 2 \tag{87}$$

c. F is true whenever

$$-4T_1 - 2T_2 + N \geq 1 \tag{88}$$

3. Use these inequalities to successively complete each column in the truth table. F will now be defined by the last column of the truth table. In this example the completed truth table is as follows:

| N | $T_1$ | $T_2$ | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 |
| 4 | 1 | 0 | 0 |
| 5 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 |

(89)

By inspection of the truth values for F it is apparent that the network realizes the odd parity function, and no further analysis is needed.

This truth table method is generally applicable to all networks of symmetric functions.

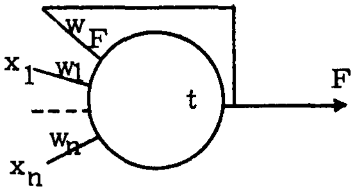
The problem of feedback loops in threshold networks will now be examined.

## Feedback in Threshold Networks

A general discussion of feedback loops in threshold networks would encompass the study of sequential logic and the theory of automata, both of which are areas outside the scope of this study. Investigation will therefore be limited to the interesting effects of feedback on a single threshold gate.

Analysis of a Single Threshold Gate with Feedback Loop. Since feedback alters the operating state of a threshold gate, the logic performed is no longer combinational but sequential in nature; and some attention must now be paid to the timing involved in the operation of threshold gates. For the simplest type of threshold element, a magnetic core, the timing is generally a two-phase synchronous system, with pulses representing the logical variables appearing at the input windings during phase one and the output pulse, if any, appearing on advent of the advance pulse during phase two. Other types of threshold gates can have the output signal appear simultaneously (for all practical purposes) with the input signals.

Consider now a threshold gate of the latter type, having a simple feedback loop with weight $w_F$. The simplifying assumption is made that the signal level representing truth is the same in both input and output circuits:



(90)

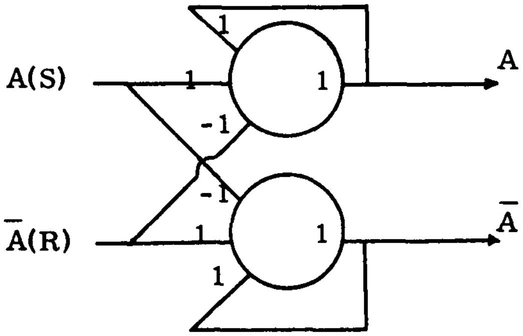
F is generated initially only when

$$\sum_{i=1}^{n} w_i x_i \geq t \text{ , } \quad x_i = 1 \text{ or } 0 \tag{91}$$

Assume that the input signals are such that an output signal appears; then there is feedback present and its effect depends on the value assigned to $w_F$. Only two limiting cases need be examined, since the feedback appears at the input simultaneously with the input signals.

a. If $w_F < t$, the feedback has no effect.

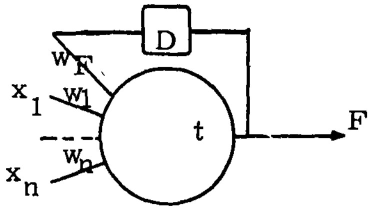
b. If $w_F \geq t$, the threshold gate locks up, i. e., a continuous output signal appears. This suggests a possible application to a very simple active threshold gate flip-flop circuit:



(92)

The functioning of the circuit is apparent by inspection.

The second case to be investigated involves feedback with a delay, say for simplicity, one clock period. Again it is assumed that the output signal, if any, appears simultaneously with the application of the input signals:



(93)

Assume now that in the initial clock period the input composite is such that an output $F_1$ appears, where $F_1$ is uniquely determined by $(w_1, \ldots, w_n; t)$. Since the feedback is delayed until the next clock period it has no immediate effect.

In the second clock period, the effect is contingent upon the value assigned to $w_F$. Several cases must be examined:

a. $w_F \geq t$: This will cause an output signal to appear regardless of the input composite; since this causes another feedback signal to appear in the succeeding clock period, the threshold gate is locked up.

b. $w_F < -(\sum_{i=1}^{n} w_i - t)$: This will cut off operation of the threshold gate gate for the current clock period; with no output signal, there will be no feedback in the succeeding clock period and the threshold gate will again be able to generate $F_1$. The net effect is that, regardless of the sequence of input composites, $F_1$ can never be generated in two successive clock periods.

c. $t > w_F \geq -(\sum_{i=1}^{n} w_i - t)$: In this range of values for $w_F$ the feedback acts in such a way as to change the effective threshold from its value t to a new value, $t - w_F$, for the next clock period. The effect of this threshold change is to generate a new function $F_2$ which is uniquely defined by $(w_1, \ldots, w_n; t - w_F)$. The sequence of states of the threshold gate can best be described in terms of the state variables Q and Q' defined as follows:

Q: the present state of the threshold gate.

Q': the next state of the threshold gate.

The input composites are defined as follows:

$(\sum wx)_1$: the input composite necessary to generate $F_1$.

$(\sum wx)_2$: the input composite necessary to generate $F_2$.

It will further be assumed for clarity that

$$(\sum wx)_1 > (\sum wx)_2 \tag{94}$$

The sequence of states can now be represented by the following "truth table" where, for Q and Q', a value of 0 denotes the state which can generate $F_1$ and a value of 1 denotes the state which can generate $F_2$:

| Q | $(\sum wx)_1$ | $(\sum wx)_2$ | F | Q' |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | X | X |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | X | X |
| 1 | 1 | 1 | 1 | 1 |

(95)

Since $(\sum wx)_1 > (\sum wx)_2$, it is obvious that whenever $(\sum wx)_1$ is true, then $(\sum wx)_2$ is also true; therefore, it is impossible to have the condition that $(\sum wx)_1 = 1$ and $(\sum wx)_2 = 0$. Hence, the symbol X is used for F and Q' for these combinations to denote a "don't care" condition. Using this table, Q' can be mapped as follows:

Q':

| | $(\sum wx)_1$ | | | |
|---|---|---|---|---|
| Q | X | 1 | 1 | 0 |
| | X | 1 | 0 | 0 |

$(\sum wx)_2$ (96)

Hence, Q' can be represented by the logical expression,

$$Q' = (\sum wx)_1 + Q(\sum wx)_2 \tag{97}$$

A state diagram can be drawn for this circuit, using the following notation:

Input composite magnitudes $(\sum wx)$ are represented by:

00: $(\sum wx) < (\sum wx)_2$ (98)

01: $(\sum wx)_2 \leq (\sum wx) < (\sum wx)_1$ (99)

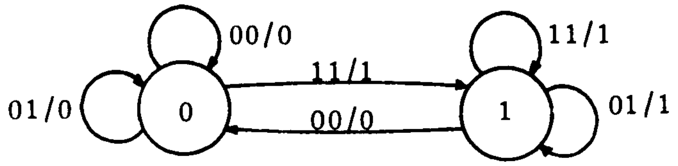
11: $(\sum wx) \geq (\sum wx)_1$ (100)

States are represented by:

0: the state which can generate $F_1$.

1: the state which can generate $F_2$

Then the state diagram is:



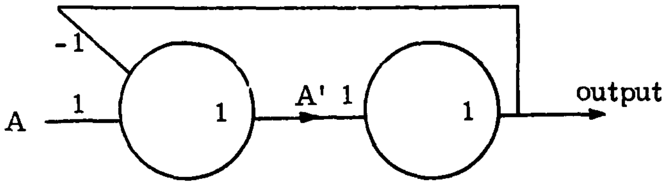
(101)

Suitable modifications of the truth table, map, expression for Q', and the state diagram are easily made if $w_F$ is so chosen that

$$(\sum wx)_1 < (\sum wx)_2 \tag{102}$$

Thus it is clear that the feedback and delay causes the threshold gate to generate one of two threshold functions in accordance with Eq (97).

This discussion of feedback loops will be concluded by examining a very simple binary counter whose operation depends on the inherent one-phase delay in a magnetic core. A delay of one full clock period is obtained by using two threshold gates in sequence:



(103)

Operation of the binary counter is demonstrated by the following truth table:

| Timing | A | A' | Output |
|---|---|---|---|
| Clock 1: $\phi_1$: | 1 | 0 | 0 |
| $\phi_2$: | 0 | 1 | 0 |
| Clock 2: $\phi_1$: | 1 | 0 | 1 |
| $\phi_2$: | 0 | 0 | 0 |
| Clock 3: $\phi_1$: | 1 | 0 | 0 |
| $\phi_2$: | 0 | 1 | 0 |
| Clock 4: $\phi_1$: | 1 | 0 | 1 |
| $\phi_2$: | 0 | 0 | 0 |

(104)

Feedback circuits with threshold gates can be used in similar fashion to obtain many other useful logic and arithmetic circuits.

## Summary

Two effective methods of analysis have been presented, one applicable to threshold networks without feedback loops, and the other to symmetric threshold gate networks without feedback loops. The effect of different types of feedback on a single threshold gate were analyzed and several simple circuits presented to emphasize the utility of feedback loops in certain applications in threshold networks. The next chapter presents a method for synthesizing a Boolean function in terms cf a network of threshold gates.

## III. Threshold Gate Synthesis of a Boolean Function from its Algebraic Expression

Conventional switching theory shows that any Boolean function can be realized by the use of AND, OR, and NOT gates, or one universal such as the Pierce or Sheffer functions. In one method, the given Boolean expression is simplified by conventional reduction methods, factored, and then decomposed into conjunctions or disjunctions of equal level terms. As an example, consider the function

$$F = A((B + C)(C + D) + KLM) \qquad (105)$$

Then

$$T_1 = (B + C) \qquad (106a)$$

$$T_2 = (C + D) \qquad (106b)$$

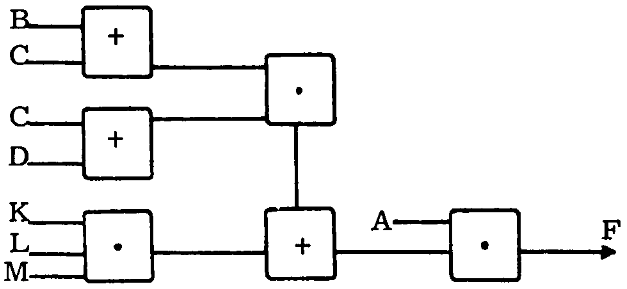
$$T_3 = T_1T_2 \qquad (106c)$$

$$T_4 = KLM \qquad (106d)$$

$$T_5 = (T_3 + T_4) \qquad (106e)$$

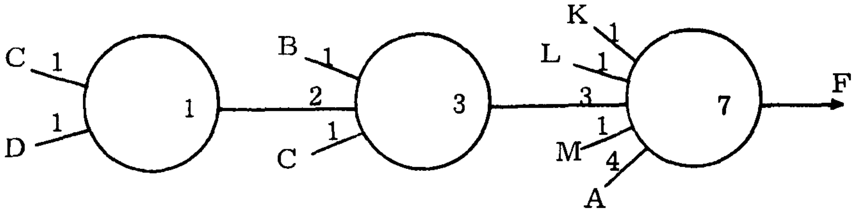
$$T_6 = F = AT_5 \qquad (106f)$$

and the function is conventionally realized with six gates:



(107)

Application of the synthesis procedure presented below leads to the following realization using three threshold gates:



(108)

The validity of the synthesis procedure rests on two theorems which, in effect, provide a means of combining conventional AND and OR functions into threshold functions. The possibility of such combination has been noted in the literature, but no synthesis procedure has been presented which utilizes this combining process. The method of synthesis is valuable in the synthesis of functions containing a large number of variables --a type of function not easily handled by any of the synthesis procedures presented in Chapter I.

Theory

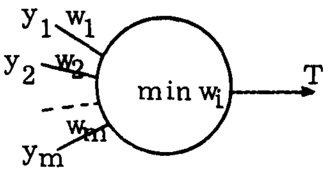
Chapter I noted that the conventional AND, OR, NOR, NOT, and NAND logic gates comprise a subset of the set of all threshold logic gates. Any normal disjunction of m binary variables, therefore, constitutes a threshold function realizable by $(w_1, w_2, \ldots, w_m; t)$ where t is given by

$$t = \min \{w_i\} \tag{109}$$

Therefore, if

$$T = y_1 + y_2 + \ldots + y_m \tag{110}$$

it is realized as shown at the top of the next page.



(111)

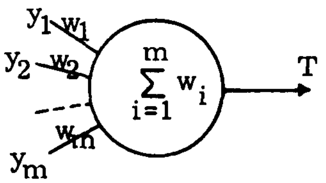
Similarly, any normal conjunction of m binary variables is also a threshold function, realized by weights and threshold ($w_1$, $w_2$, . . . , $w_m$;t) where t is given by

$$t = \sum_{i=1}^{m} w_i \qquad (112)$$

Therefore, if

$$T = y_1 y_2 . . . y_m \qquad (113)$$

it is realized as shown:



(114)

Consider now a function $T_2$ consisting of $T_1$ and n other binary variables, in one of the following forms:

$$T_2 = T_1 + x_1 + x_2 + . . . + x_n \qquad (115a)$$

or

$$T_2 = T_1 x_1 x_2 . . . x_n \qquad (115b)$$

where $T_1$ is a threshold function of m binary variables. Then $T_2$ can be realized by a single threshold gate, which is, in effect, a modification of the threshold gate used to realize $T_1$. The basis for such modification is given in the two theorems stated below.

Theorem 1. Let $T_1$ be a threshold function of m binary variables, realized by weights $w_1, w_2, \ldots, w_m$ and threshold $t_1$; consider now a second threshold function $T_2$, a disjunction of $T_1$ and n binary variables such that

$$T_2 = T_1 + x_1 + x_2 + \ldots + x_n \qquad (115a)$$

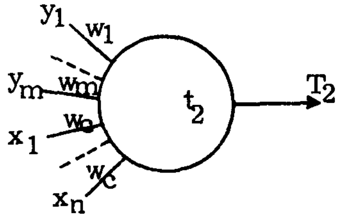
Then $T_2$ can be realized by a single threshold gate using the same weights $w_1, w_2, \ldots, w_m$ for the m original inputs of $T_1$; a common weight $w_c$, given by

$$w_c = t_1 \qquad (116)$$

for the additional n inputs to $T_2$; and a threshold $t_2$ given by

$$t_2 = t_1 \qquad (117)$$

Proof by Construction: 1. The proposed connection is as follows:



(118)

2. For an assignment of truth values to the m original inputs of $T_1$ such that $T_1$ is true, or if any x is true, then $T_2$ is true. This requires that

$$\sum_{i=1}^{m} w_i y_i \geq t_2 \quad \text{when } T_1 \text{ is true,} \qquad (119)$$

and

$$w_c \geq t_2 \tag{120}$$

3. We are free to select any values for $w_c$ and $t_2$ consistent with Ineqs (119) and (120). Therefore, select

$$w_c = t_1 \tag{116}$$

and

$$t_2 = t_1 \tag{117}$$

and the threshold function $T_2$ is realized.

<u>Theorem 2</u>: Let $T_1$ be defined as in Theorem 1; consider now a second threshold function, $T_2$, a conjunction of $T_1$ and n binary variables such that

$$T_2 = T_1 x_1 x_2 \ldots x_n \tag{115b}$$

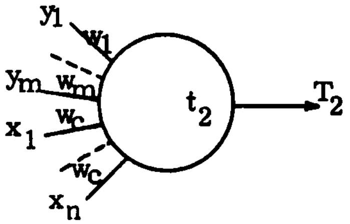
Then $T_2$ can be realized by a single threshold gate using the weights $w_1, w_2, \ldots, w_m$ for the m original inputs of $T_1$; a common weight $w_c$ given by

$$w_c = \sum_{i=1}^{m} w_i - t_1 + 1 \tag{121}$$

for the n additional inputs to $T_2$; and a threshold $t_2$ given by

$$t_2 = t_1 + nw_c \tag{122}$$

<u>Proof by Construction:</u> 1. The proposed connection is as follows:



(123)

2. For an assignment of truth values to the m inputs of $T_1$ such that $T_1$ is true, and if all x's are true, then $T_2$ is true. This requires that

$$t_2 \leq t_1 + nw_c \tag{124}$$

3. If all y's are true and at least one x is false, then $T_2$ is false. Hence,

$$t_2 > \sum_{i=1}^{m} w_i + (n - 1)w_c \tag{125}$$

4. Combining Ineqs (124) and (125),

$$\sum_{i=1}^{m} w_i + (n - 1)w_c < t_2 \leq t_1 + nw_c$$

or

$$\sum_{i=1}^{m} w_i - w_c < t_2 - nw_c \leq t_1 \tag{126}$$

5. We are free to select any values for $w_c$ and $t_2$ consistent with Ineq (126). Therefore, select $w_c$ and $t_2$ such that

$$\sum_{i=1}^{m} w_i - w_c + 1 = t_2 - nw_c = t_1$$

or,

$$w_c = \sum_{i=1}^{m} w_i - t_1 + 1 \tag{121}$$

and

$$t_2 = t_1 + nw_c \tag{122}$$

and the threshold function $T_2$ is realized. These results are now used as the basis of the following synthesis procedure.

Procedure

1. Simplify and reduce the given Boolean function using normal switching theory techniques.

2. Determine the threshold function order of synthesis as follows:

a. Insert parentheses (if not already present) around each term or group of terms that correspond to a conventional AND or OR gate realization in normal switching theory synthesis. In the example of Eq (105), parentheses are inserted, where needed, around those terms corresponding to $T_1$, $T_2, \ldots,$ $T_6$, i. e., Eqs (106). Thus, Eq (105) would appear as

$$F = (A((B + C)(C + D) + (KLM))) \tag{105}$$

b. Number opening parentheses increasing from the left; number closing parentheses decreasing from the left; and at adjacent parentheses of opposite type, leave the numbers unchanged:

$$F = \overset{1}{(}A\overset{2}{(}\overset{3}{(}B + C\overset{3}{)}\overset{3}{(}C + D\overset{3}{)} + \overset{3}{(}KLM\overset{3}{)}\overset{2}{)}\overset{1}{)} \tag{105}$$

c. The order of synthesis is from the highest order terms to the lst order term representing the complete function. Referring to Eq (105), the 3rd order terms are realized first, then the 2nd order term, and then the lst order term.

3. Sub-functions to be realized ($T_1, \ldots,$ $T_6$ of Eq 106 for example) will be in one of the following forms:

a. Simple AND or OR functions: These are realized using a common weight of l, and a threshold of $\sum_{i=1}^{m} w_i$ or l, respectively.

b. Functions of the forms given by Eqs (115): These are realized by applying Theorems 1 and 2 respectively.

c. Functions of the form $T_3 = T_1T_2$: A choice exists as to whether to apply Theorem 2 to $T_1$ or $T_2$. With one exception, explained below, Theorem 2 is always applied to the T having the least sum of weights plus threshold. This leads to a lower aggregate sum of weights and thresholds in the overall realization. The exception to this rule comes only when one function, say $T_1$, is a simple (i. e., has equal weights) conjunctive realization, and $T_2$ is a disjunctive realization. Then $T_3$ is expressed as

$$T_3 = T_1T_2 = y_1y_2 \ldots y_mT_2 \tag{127}$$

and Theorem 2 is applied to $T_2$, resulting in the elimination of gate $T_1$. See Eq (134) in the example to follow.

d. Functions of the form $T_3 = T_1 + T_2$: Again, a choice exists as to whether to apply Theorem 1 to $T_1$ or to $T_2$. With one exception, Theorem 1 is always applied to the T having the least sum of weights plus threshold. The exception to this rule comes when one function, say $T_1$, is a simple (i. e., has equal weights) disjunctive realization, and $T_2$ is a conjunctive realization. Then $T_3$ is expressed as

$$T_3 = T_1 + T_2 = y_1 + y_2 + \ldots + y_m + T_2 \tag{128}$$

and Theorem 1 is applied to $T_2$, resulting in the elimination of gate $T_1$. The following example illustrates the complete synthesis procedure.

Example. Given the function

$$F = I(AB + CD(E + G + H)$$
$$+ JKL(M(N + O) + (O + P)(QR(S + T + U) + V) + WXY + Z)) \tag{129}$$

1. Insert parentheses and determine order:

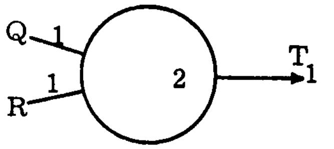
$$F = \overset{1}{(}I\overset{2}{(}\overset{3}{(}AB\overset{3}{)} + \overset{3}{(}\overset{4}{(}CD\overset{4}{)}\overset{4}{(}E + G + H\overset{4}{)}\overset{3}{)} + \overset{3}{(}\overset{4}{(}JKL\overset{4}{)}\overset{4}{(}\overset{5}{(}M\overset{6}{(}N + O\overset{6}{)}\overset{5}{)}$$

$$+ \overset{5}{(}\overset{6}{(}O + P\overset{6}{)}\overset{6}{(}\overset{7}{(}\overset{8}{(}QR\overset{8}{)}\overset{8}{(}S + T + U\overset{8}{)}\overset{7}{)} + V\overset{6}{)}\overset{5}{)} + \overset{5}{(}WXY\overset{5}{)} + Z\overset{4}{)}\overset{3}{)}\overset{2}{)}\overset{1}{)} \quad (129)$$

Note that unsubscripted T is used as one of the binary variables.

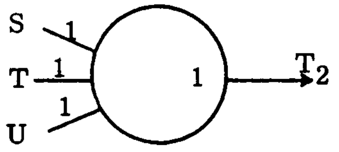
2. Realize 8th order terms:

$T_1 = QR$:



(130)

$T_2 = S + T + U$:



(131)

3. Realize 7th order terms:

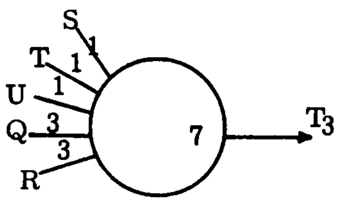
$T_3 = T_1T_2$: Express $T_3$ as $T_3 = QRT_2$ and apply Th 2 to $T_2$:

$$^*w_3 = \sum_{i=1} w_{2i} - t_2 + 1$$
$$= 3 - 1 + 1 = \underline{3} \quad (132)$$

and

$$t_3 = t_2 + nw_3$$
$$= 1 + (2)(3) = \underline{7} \quad (133)$$

Thus $T_3$ is realized by:

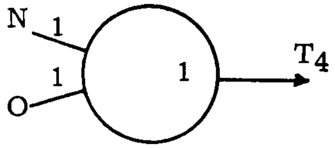


(134)

*Subscripts on w and t refer to a particular threshold function $T_j$.

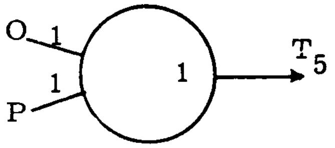
4. Realize 6th order terms:

$T_4 = N + O$:



(135)

$T_5 = O + P$:



(136)

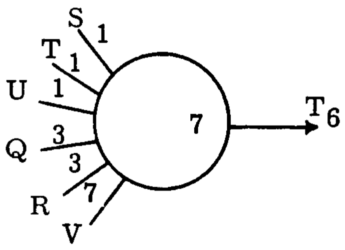
$T_6 = T_3 + V$: Apply Th 1 to $T_3$:

$$w_6 = t_3 = \underline{7} \qquad (137)$$

and

$$t_6 = t_3 = \underline{7} \qquad (138)$$

Thus $T_6$ is realized by:



(139)

5. Realize 5th order terms:

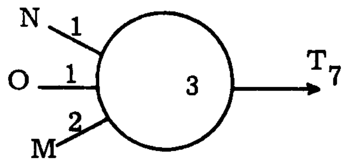
$T_7 = MT_4$: Apply Th 2 to $T_4$:

$$w_7 = \sum_{i=1}^{m} w_{4i} - t_4 + 1$$
$$= 2 - 1 + 1 = \underline{2} \qquad (140)$$

and

$$t_7 = t_4 + nw_7$$
$$= 1 + (1)(2) = \underline{3} \qquad (141)$$

Thus $T_7$ is realized by:



(142)

$T_8 = T_5T_6$: Apply Th 2 to $T_5$:

$$w_8 = \sum_{i=1}^{m} w_{5i} - t_5 + 1$$

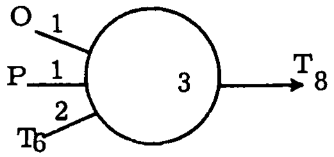
$$= 2 - 1 + 1 = \underline{2} \qquad (143)$$

and

$$t_8 = t_5 + nw_8 \qquad (144)$$

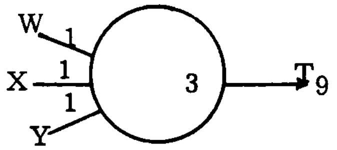
$$= 1 + (1)(2) = \underline{3}$$

Thus $T_8$ is realized by:



(145)

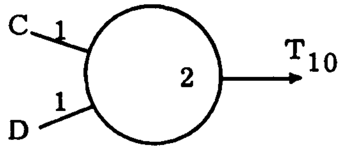
$T_9$ = WXY:



(146)

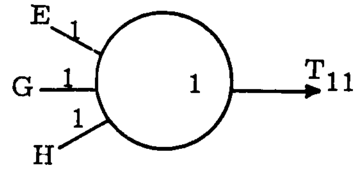
6. Realize 4th order terms:

$T_{10}$ = CD:



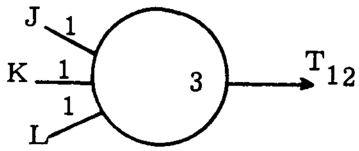
(147)

$T_{11} = E + G + H$:



(148)

$T_{12} = JKL$:



(149)

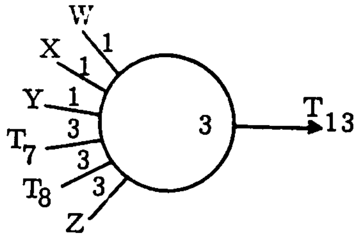
$T_{13} = T_7 + T_8 + T_9 + Z$: Apply Th 1 to $T_9$:

$$w_{13} = t_9 = \underline{3} \tag{150}$$

and

$$t_{13} = t_9 = \underline{3} \tag{151}$$

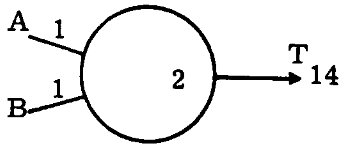
Thus $T_{13}$ is realized by:



(152)

7. Realize 3rd order terms:

$T_{14} = AB$:



(153)

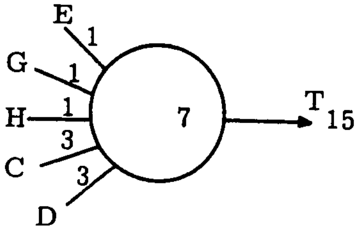
$T_{15} = T_{10}T_{11}$: Express $T_{15}$ as $T_{15} = CDT_{11}$; apply Th 2 to $T_{11}$:

$$w_{15} = \sum_{i=1}^{m} w_{11i} - t_{11} + 1$$
$$= 3 - 1 + 1 = \underline{3} \tag{154}$$

and

$$t_{15} = t_{11} + nw_{15}$$
$$= 1 + (2)(3) = \underline{7} \qquad (155)$$

Thus $T_{15}$ is realized by:



(156)

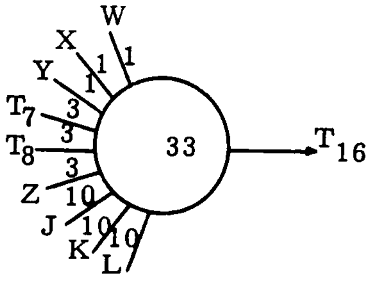
$T_{16} = T_{12}\ T_{13}$: Apply Th 2 to $T_{13}$:

$$w_{16} = \sum_{i=1}^{m} w_{13i} - t_{13} + 1$$
$$= 12 - 3 + 1 = \underline{10} \qquad (157)$$

and

$$t_{16} = t_{13} + nw_{16} \qquad (158)$$
$$= 3 + (3)(10) = \underline{33}$$

Thus $T_{16}$ is realized by:



(159)

8. Realize 2nd order term:

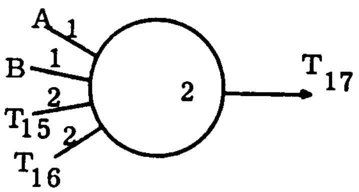
$T_{17} = T_{14} + T_{15} + T_{16}$: Apply Th 1 to $T_{14}$:

$$w_{17} = t_{14} = \underline{2} \qquad (160)$$

and

$$t_{17} = t_{14} = \underline{2} \qquad (161)$$

Thus $T_{17}$ is realized by:



(162)

9. Realize 1st order term:

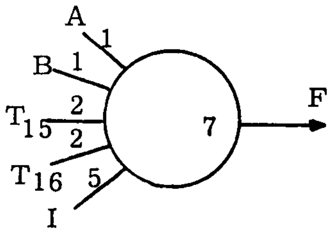
$F = IT_{17}$: Apply Th 2 to $T_{17}$:

$$w_{18} = \sum_{i=1}^{m} w_{17} - t_{17} + 1$$
$$= 6 - 2 + 1 = \underline{5} \qquad (163)$$

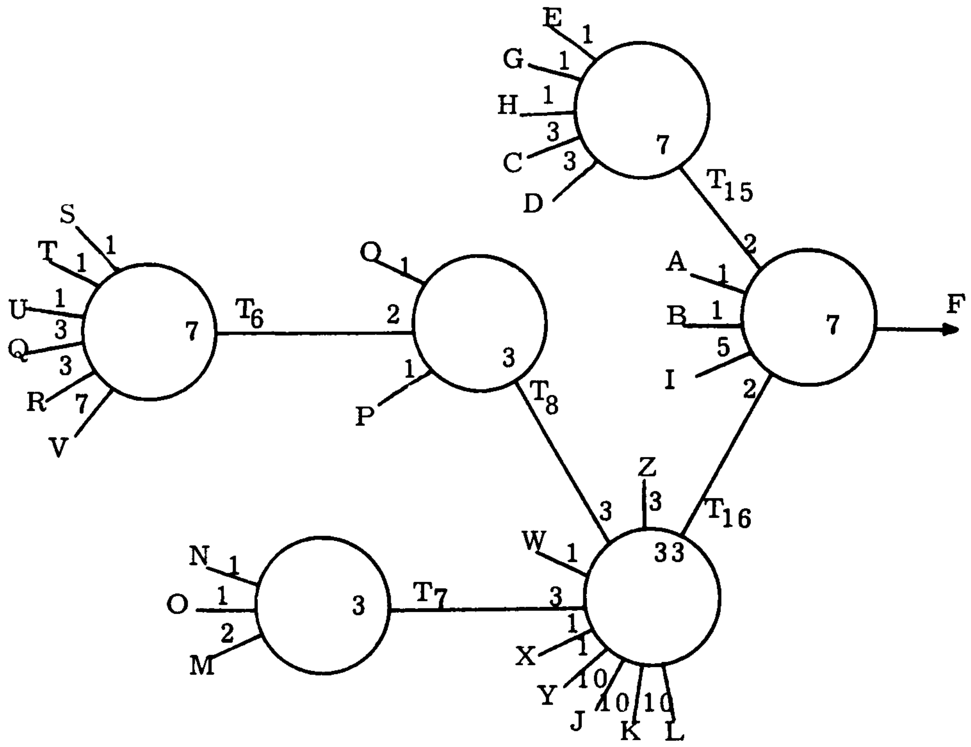
and

$$t_{18} = t_{17} + nw_{18}$$
$$= 2 + (1)(5) = \underline{7} \qquad (164)$$

Thus F is realized by: (165)



10. Reconstruct the function:



(166)

Thus, F is realized using six threshold gates; in contrast, a conventional switching theory realization would require eighteen AND and OR gates. No threshold gate has an unreasonable sum of input weights, since threshold elements are presently available which are capable of proper discrimination with any input combination of weights totaling up to 51*.

Note that the manner in which the function was originally factored directly affects the final form of the realization.

## Conclusion

The procedure given allows a complex Boolean function to be synthesized using threshold gates. The method is rapid, suitable for hand calculation, applicable to all switching functions, and yields a reasonable (in terms of weights, thresholds, and number of elements) realization of the function. The example contained twenty-five variables, far exceeding the number that can be handled by any procedure outlined in Chapter I, unless these procedures are programmed on a digital computer.

The primary disadvantage of the procedure is that the threshold net produced is not necessarily minimal. However, it does provide a starting point for further efforts at reduction, either by heuristic reasoning or other techniques to be developed.

*Learned in conversation with Dr. S. B. Akers, General Electric Laboratories, Syracuse, New York.

## IV. Counter Example to Stuart's Algorithm

An algorithm to test for linear separability of a function has been presented by Captain R. B. Stuart (Ref 26). If the function is proved to be linearly separable the algorithm provides a set of weights and a threshold which realize the given function. Application of the method to many known threshold functions supported Stuart's conjecture that the algorithm constituted a necessary and sufficient test for linear separability. However, investigation of the algorithm's validity provided a counter example to disprove the conjecture. The algorithm and counter example are presented below after first developing the matrix notation to be used.

### Notation

Valuations on the n variables of a Boolean function F which make F true are termed "true valuations;" all valuations not true valuations are termed "false valuations." Rows of F's truth table corresponding to true (false) valuations will be called "true (false) rows."

Consider now a Boolean function F of m minterms: Permute the rows of F's truth table so as to have the first m rows of the truth table be the m true rows. This truth table is called the "altered truth table." Define a truth table matrix P whose elements are those of the altered truth table; partition this matrix so that the m true rows form the sub-matrix $P_T$ and the false rows form the sub-matrix $P_F$:

$$P = \left[\begin{array}{c} P_T \\ \hdashline P_F \end{array}\right] \tag{167}$$

Then the function vector $\underline{f}$ becomes

$$\underline{f} = \begin{bmatrix} \underline{1} \\ \hdashline \underline{0} \end{bmatrix} \tag{168}$$

Now map P into a new matrix defined by

$$Q = \begin{bmatrix} Q_T \\ \hdashline Q_F \end{bmatrix} \quad \text{where} \quad \begin{array}{l} q_{ij} = 1 \text{ if } p_{ij} = 1 \\ q_{ij} = -1 \text{ if } p_{ij} = 0 \end{array} \tag{169}$$

Define the weight vector $\underline{W}^T$ (superscript T indicating transpose),

$$\underline{W}^T = \underline{1}^T Q_T \tag{170}$$

i. e.,

$$w_j = \underline{1}^T \underline{q}_j \tag{171}$$

where $q_j$ is the jth column of Q.

Define the weighted minterm vector $\underline{g}$,

$$\underline{g} = P\underline{W} = P\begin{bmatrix} Q_T \end{bmatrix}^T \underline{1} \tag{172}$$

i. e.,

$$g_i = \underline{p}_i \underline{W} \tag{173}$$

Then

$$\underline{g} = \begin{bmatrix} g_T \\ \hdashline g_F \end{bmatrix} = \begin{bmatrix} P_T \\ \hdashline P_F \end{bmatrix} \begin{bmatrix} Q_T \end{bmatrix}^T \underline{1} \tag{174}$$

In effect the above matrix manipulation accomplishes the following:

The truth table representing the given function is so arranged that the first m rows (which comprise $P_T$) are true rows which correspond to the minterms of F; and the remaining $2^n - m$ rows (which comprise $P_F$) are false rows which correspond to the minterms of $\bar{F}$. All 0's in $P_T$ are then changed to -1's to form the sub-matrix $Q_T$ and each column of $Q_T$ is summed. Each column sum gives the weight $w_j$ to be assigned to the variable represented by that column. The row vector $\underline{W}^T$ contains these weights for all variables. Each minterm of the n variables is then assigned a weight $g_i$ equal to the sum of the individual weights of the uncomplemented variables in the particular minterm; then $\underline{g}$ contains the weights for all minterms. The algorithm can now be presented.

<u>Stuart's Algorithm</u>

Given an arbitrary Boolean function F of m minterms:

1. Form

$$\underline{g} = \begin{bmatrix} \underline{g}_T \\ \hdashline \underline{g}_F \end{bmatrix} \tag{175}$$

2. Test for linear separability:

$$\text{Is } \min\left\{\underline{g}_T\right\} > \max\left\{\underline{g}_F\right\} \text{ ?} \tag{176}$$

If not, F is not a linearly separable function. If so, F is linearly separable and can be realized by weights $\underline{W}^T$ and threshold t where

$$t = \min\left\{g_T\right\} \tag{177}$$

This test can be conveniently applied ( at least for $n \leq 6$) by using Karnaugh maps as described in the following procedure.

Procedure. 1. Plot the given function on a Karnaugh map. Note that each entry on the map represents a minterm of the n variables.

2. Form the partial truth table $P_T$ from the m minterms of the given function.

3. Change each 0 to -1 in $P_T$ to obtain $Q_T$.

4. Sum each column of $Q_T$ to obtain the weight $w_j$ to be assigned to the corresponding variable.

5. Weight each minterm of the n variables by summing the weights of the uncomplemented variables in the minterm.

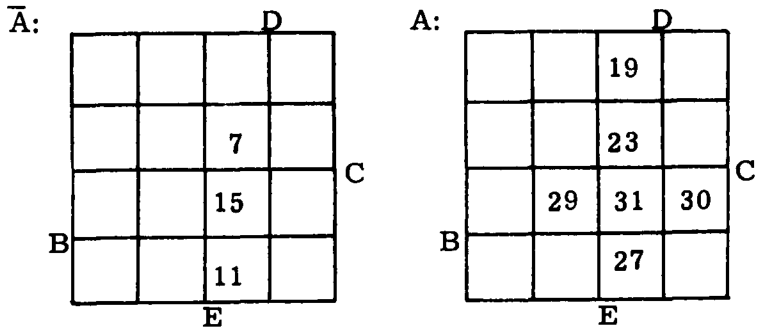
6. Enter the minterm weights on a Karnaugh map and circle those entries representing the weighted minterms of F.

7. Determine by inspection of the map if all weighted minterms of F are greater than all weighted minterms of $\bar{F}$. If not, the function is not linearly separable; if so, then the function is linearly separable and can be realized by the weights determined in step 4 above and a threshold t equal to the minimum weighted minterm of F.

Example 1. Given the function

$$F = DE(A + B + C) + ABC(D + E) \qquad (178)$$

1. Plot F on a Karnaugh map:



(179)

The numbers on the map are the conventional truth table designations for the rows corresponding to the particular minterm.

2. Form $P_T$. Note that subtracting the number of 0's in each column from the number of 1's in each column gives the same weight as would be obtained by forming $Q_T$ and then summing the columns to determine the weights for each variable:

$P_T$:

| # | A | B | C | D | E |
|---|---|---|---|---|---|
| 7 | 0 | 0 | 1 | 1 | 1 |
| 11 | 0 | 1 | 0 | 1 | 1 |
| 15 | 0 | 1 | 1 | 1 | 1 |
| 19 | 1 | 0 | 0 | 1 | 1 |
| 23 | 1 | 0 | 1 | 1 | 1 |
| 27 | 1 | 1 | 0 | 1 | 1 |
| 29 | 1 | 1 | 1 | 0 | 1 |
| 30 | 1 | 1 | 1 | 1 | 0 |
| 31 | 1 | 1 | 1 | 1 | 1 |
| $w_j$ | 3 | 3 | 3 | 7 | 7 |

(180)

3. Using the weights just determined, form the weighted minterms and enter them on a Karnaugh map. Circle those minterms of F:

| | | | |
|---|---|---|---|
| 0 | 7 | 14 | 7 |
| 3 | 10 | 17 | 10 |
| 6 | 13 | 20 | 13 |
| 3 | 10 | 17 | 10 |

| | | | |
|---|---|---|---|
| 3 | 10 | 17 | 10 |
| 6 | 13 | 20 | 13 |
| 9 | 16 | 23 | 16 |
| 6 | 13 | 20 | 13 |

(181)

By inspection, since

$$\min\{g_T\} = 16 > \max\{g_F\} = 14 \qquad (182)$$

the function is linearly separable and can be realized by

$$\underline{W}^T = (3, 3, 3, 7, 7) \qquad (183)$$

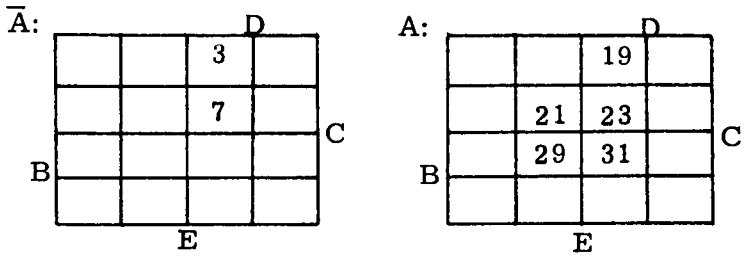
and

$$t = 16 \qquad (184)$$

Example 2. Given the function

$$F = E(AC + \bar{B}D) \qquad (185)$$

1. Plot F on a Karnaugh map:



(186)

2. Form $P_T$ and find the weights:

$P_T$:

| # | A | B | C | D | E |
|---|---|---|---|---|---|
| 3 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 1 | 1 | 1 |
| 19 | 1 | 0 | 0 | 1 | 1 |
| 21 | 1 | 0 | 1 | 0 | 1 |
| 23 | 1 | 0 | 1 | 1 | 1 |
| 29 | 1 | 1 | 1 | 0 | 1 |
| 31 | 1 | 1 | 1 | 1 | 1 |
| $w_j$ | 3 | -3 | 3 | 3 | 7 |

(187)

3. Using the weights just determined, form the weighted minterms and enter them on a Karnaugh map. Circle the minterms of F:

| | | | |
|---|---|---|---|
| 0 | 7 | 10 | 3 |
| 3 | 10 | 13 | 6 |
| 0 | 7 | 10 | 3 |
| -3 | 4 | 7 | 0 |

| | | | |
|---|---|---|---|
| 3 | 10 | 13 | 6 |
| 6 | 13 | 16 | 9 |
| 3 | 10 | 13 | 6 |
| 0 | 7 | 10 | 3 |

(188)

By inspection note that

$$\min\{g_T\} \ngtr \max\{g_F\} \qquad (189)$$

and therefore the function is not linearly separable.

In both examples the algorithm correctly determined if the function was linearly separable; but a counter example will now be presented to disprove the algorithm.

Counter Example

Given the Boolean function

$$F = ABC + D(A + B + C) \qquad (190)$$

1. Plot the function on a Karnaugh map:

| | | | |
|---|---|---|---|
| | | 3 | |
| | 5 | 7 | |
| | 13 | 15 | 8 |
| | 9 | 11 | |

(191)

2. Form $P_T$ and determine the weights:

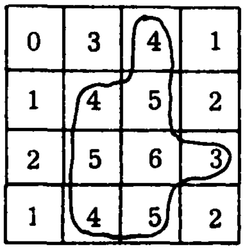
$P_T$:

| # | A | B | C | D |
|---|---|---|---|---|
| 3 | 0 | 0 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 |
| 9 | 1 | 0 | 0 | 1 |
| 11 | 1 | 0 | 1 | 1 |
| 13 | 1 | 1 | 0 | 1 |
| 14 | 1 | 1 | 1 | 0 |
| 15 | 1 | 1 | 1 | 1 |
| $w_j$ | 2 | 2 | 2 | 6 |
| $w_j/2$ | 1 | 1 | 1 | 3 |

(192)

Note that the weights can be reduced by dividing through by two.

3. Form the weighted minterms and enter them on a Karnaugh map. Circle the minterms of F:



| | | | |
|---|---|---|---|
| 0 | 3 | 4 | 1 |
| 1 | 4 | 5 | 2 |
| 2 | 5 | 6 | 3 |
| 1 | 4 | 5 | 2 |

(193)

By inspection, note that

$$\min\left\{\underline{g}_T\right\} \not> \max\left\{\underline{g}_F\right\} \tag{194}$$

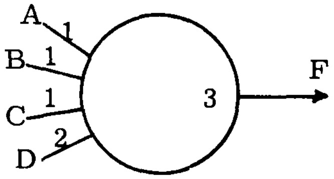
Therefore, according to the algorithm, the function is not linearly separable. But F is indeed a threshold function and it can be realized by

$$\underline{W}^T = (1, 1, 1, 2) \qquad (195)$$

and a threshold

$$t = 3 \qquad (196)$$

This realization is shown below:



(197)

Hence Stuart's algorithm does not provide a necessary test for linear separability.

The algorithm has a limited utility in that it does provide a sufficient test for linear separability. Truth of this statement results from the defining inequalities for a threshold function: If any set of n + 1 constants can be found which satisfy these inequalities, then the function is linearly separable. Since the algorithm generates such a set of n + 1 constants for those functions it identifies as being linearly separable, it does constitute a sufficient test.

## Conclusions

The practical utility of the algorithm is limited by the following considerations:

a. The algorithm leaves open the question of the realizability of those functions it fails to identify as being linearly separable.

b. The realizations generated for those functions identified as threshold functions are, in general, not minimal.

c. The algorithm is difficult to apply to functions of more than six variables.

It is interesting to note that tables have now been compiled which list all linearly separable functions, and the weights and thresholds to realize them, for all functions of up to six variables (Ref 30). An added advantage of the table is that the values of the weights and thresholds are minimal values.

## V. Conclusions and Recommendations

### Conclusions

Threshold logic now comprises a useful and growing segment of computer technology. It is proving to be of great value in the development of more sophisticated computers and "thinking" machines; and it is providing simpler and more economical realizations for conventional logic problems. Threshold elements have many practical physical realizations, and improved devices are being developed.

Certain improvements in threshold logic are needed, the most important of which are the following:

a. an algebraic test for linear separability;

b. a method to systematically eliminate redundant elements in complex threshold networks;

c. a method for reducing (to minimal values) the weights and thresholds of complex threshold function realizations;

d. simpler methods for decomposing Boolean functions into threshold functions, and tests to determine if the decomposition is optimal;

e. better synthesis methods for functions of a large number of variables; and

f. better procedures for including the effects of physical constraints on synthesis procedures.

With the widespread interest in threshold logic now existing, it is evident that many of these problems will soon be solved.

## Recommendations for Further Study

The partial list of needed improvements in the field of threshold logic, given above, indicates many areas for interesting and useful investigation. Specifically, the following subjects are felt to merit consideration as possible thesis topics:

a. A more detailed study of feedback in threshold gate networks would provide useful design procedures and a better understanding of the potential benefits of feedback in sequential threshold logic design.

b. An algebraic test for linear separability would be a valuable contribution.

c. Methods for decomposing Boolean functions into a minimal number of threshold functions are urgently needed. Several methods exist in the literature, but they give no assurance of a minimal circuit realization of the function.

d. The synthesis procedure outlined in Chapter III could be refined to provide ways of accomplishing further simplification of the resulting threshold gate realization.

e. The application of non-linear signal flow graph techniques to the analysis of threshold networks would provide a clearer insight into the characteristics of these networks.

f. To increase the utility of existing synthesis procedures, the effects of physical constraints on these procedures could be further investigated.

g. Improved synthesis procedures can undoubtedly be developed with further research; and computer programs for such procedures would be very useful.

## Bibliography

1. Akers, S. B., Jr. "A Truth Table Method for the Synthesis of Combinational Logic." *IRE Transactions on Electronic Computers*, EC-10:604-614 (December 1961).

2. -----. *Threshold Logic and Two-Person Zero-Sum Games*. Proceedings of the Second Annual AIEE Symposium on Switching Circuit Theory and Logical Design (September 1961).

3. -----, and T. C. Robbins. *Logical Design With Three-Input Majority Gates*. General Electric Technical Information Series R61 ELS-141 (March 1962).

4. -----. "On a Theory of Boolean Functions." *Journal of the Society of Industrial and Applied Mathematics*, 7:487-498 (December 1959).

5. Ausfresser, H. D. *The Synthesis of Boolean Functions with Threshold Logic*. Westinghouse Electric Corporation, Air Armament Division, Report No. AA-2424 (August 1961).

6. Caldwell, J. H. *Switching Circuits and Logical Design*. New York: John Wiley and Sons, Inc., 1960.

7. Chow, C. K. *On the Characterization of Threshold Functions*. Proceedings of the Second Annual AIEE Symposium on Switching Circuit Theory and Logical Design (September 1961).

8. Coates, C. L. et al. "A Simplified Procedure for the Realization of Linearly-Separable Switching Functions." *IRE Transactions on Electronic Computers*, EC-11:447-459 (August 1962).

9. Cohn, M. and R. Lindaman. "Axiomatic Majority Decision Logic." *IRE Transactions on Electronic Computers*, EC-10:17-21 (March 1961).

10. Einhorn, S. N. "The Use of the Simplex Algorithm in the Mechanization of Boolean Functions by Means of Magnetic Cores." *IRE Transactions on Electronic Computers*, EC-10:615-622 (December 1961).

11. Fischler, M. A. and E. A. Poe. "Threshold Realization of Arithmetic Circuits." *IRE Transactions on Electronic Ccmputers*, EC-11:287-288 (April 1962).

12. Gabelman, I. J. *Synthesis of Boolean Functions Using a Single Threshold Element*. Rome Air Development Center, RADC-TDR 62-99 (March 1962).

13. -----. "A Note on the Realization of Boolean Functions Using a Single Threshold Element." Proceedings of the IRE, 50:225-226 (February 1962).

14. Highleyman, W. H. "A Note on Linear Separation." IRE Transactions on Electronic Computers, EC-10:777 (December 1961).

15. Hohn, F. E. Applied Boolean Algebra. New York: The Macmillan Co., 1961.

16. Hu, S. T. Linearly Separable Switching Functions. Lockheed Aircraft Co., Lockheed Missile and Space Division Technical Document LMSD-703024, Sunnyvale, California (July 1960).

17. Karnaugh, M. "Pulse Switching Circuits Using Magnetic Cores." Proceedings of the IRE, 43:570-584 (May 1955).

18. Kautz, W. H. "Realization of Symmetric Switching Functions with Linear-Input Logical Elements." IRE Transactions on Electronic Computers, EC-10:371-378 (September 1961).

19. McNaughton, R. "Unate Truth Functions." IRE Transactions on Electronic Computers, EC-10:1-5 (March 1961).

20. Minnick, R. C. "Linear Input Logic." IRE Transactions on Electronic Computers, EC-10:6-16 (March 1961).

21. Muroga, S., et al. "Theory of Majority Decision Elements." Journal of the Franklin Institute, Vol 271 (1961).

22. Paull, M. C. and E. J. McCluskey, Jr. "Boolean Functions Realizable with Single Threshold Devices." Proceedings of the IRE, 48:1335-1337 (July 1960).

23. Phister, M., Jr. Logical Design of Digital Computers. New York: John Wiley and Sons, Inc., 1958.

24. Scott, N. R. Analog and Digital Computer Technology. New York: McGraw Hill Book Co. Inc., 1960.

25. Stram, O. B. "Arbitrary Boolean Functions of n Variables Realizable in Terms of Threshold Devices." Proceedings of the IRE, 49:210-220 (January 1961).

26. Stuart, R. B. Thresholding in Self-Organizing Machines. Master's Thesis, USAF Institute of Technology, GA/EE/62-3.

27. Tanaka, R. I., *et al*. *Investigation of Threshold Switching Techniques*. ASD Technical Report ASD TDR-62-308.

28. Winder, R. O. *Single Stage Threshold Logic*. Proceedings of the First Annual AIEE Symposium on Switching Circuit Theory and Logical Design (October 1960).

29. -----. *More About Threshold Logic.* Proceedings of the Second Annual AIEE Symposium on Switching Circuit Theory and Logical Design (September 1961).

30. -----. *Threshold Logic*. Doctoral Dissertation, Princeton University (May 1962).